به نام خدا

جلسه اول

سیستم عامل:

☐ نرم افزاری است که کامپیوتر را کنترل می کند.

☐ نرم افزاری است که سخت افزار را برای کاربر، قابل استفاده می کند.

سرویس ها ی سیستم عامل در سه سطح قرار دارند

aplication service −3 library service −2 kernel Service −1

هسته سیستم عامل **kernel** میباشد که در حد ممتاز اجرا می شود.سیستم حداقل از دو حالت اجرا حمایت می کند

1-حالت ممتاز (حق بالا) 2-حالت عادی(حق پایین)

توضیحات اضافی:حالت غیر ممتاز (حق پایین) حالتی است که اجازه اجرای دستورالعمل های حساس سخت افزاری مثل دستورالعمل توقف و دستورالعمل های ورودی /خروجی را نمی دهند، این حالت را حالت کاربر نیز گویندچرا که برنامه های کاربران معمولا در این حالت اجرا می شوند حالت ممتاز یک وضعیت اجرائی است که به تمام دستورالعمل های سخت افزار اجازه اجرا می دهد، که به این حالت حالت سیستم ،حالت کنترل یا حالت هسته می گویند

Kernel یا هسته سیستم:

1−به وقفه واکنش نشان می دهد

2−به سخت افزار دسترسی مستقیم دارد

3−مقیم دائمی حافظه میباشد

4−ایجاد پردازه،ختم پردازه ،زمانبندی و ... در قسمت Kernel انجام میشود.

اهداف سیستم عامل:

1- مدیریت منابع مانند پردازنده ،حافظه، گذرگاهها،تایمرهاو ...

2- ماشین توسعه یافته(**extend machine**)

3- ایجاد لایه تجرید: مجموعه برنامه هائی است که جزئیات سخت افزاری را از دید کاربر مخفی نگه می دارد

4- ایجاد واسط کاربری قوی و خوش تعریف

تاریخچه سیستم عامل:جهت بررسی تاریخچه سیستم عامل می بایست تاریخچه معماری ماشین هائی رادر نظر گرفت که سیستم عامل ها بر روی آنها اجرا د

1- نسل اول 55-1945:در ساخت کامپیوترها از لامپ خلا استفاده می شد-ماشین های آن موقع قادر بودند در هر لحظه یک بیت را اجرا کنند-زبان های برنامه نویسی حتی اسمبلی ناشناخته بودند-پس چیزی به نام سیستم عامل وجود نداشت.

2- نسل دوم 65-1955 : کامپیوترها از ترانزیستور ساخته شدند-شرکت جنرال سیستم اولین سیستم عامل رابرای کامپیوتر IBM 701 نوشت-سیستم عامل ها به صورت batch کار می کردند.

3- نسل سوم80 - 1965 :پیدایش مدارات مجتمع IC-پیدایش multiprograming- پیدایش vlsi-lsi ؛این دوره معادل میشود با عرضه سیستم عامل برای کامپیوترهای شخصی، همچنین سیستم عامل ها روی شبکه کار می کردند،معمولا اولین سیستم عامل هائی که در این دوره به برتری دست یافتند می توان به Ms Dos برای خانواده کامپیوتری Intel و Unix برای کامپیوترهای motorola ، اشاره کرد.

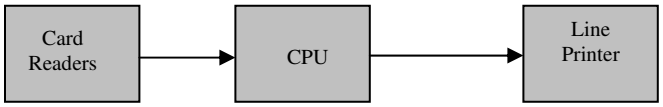
سیستم ها از جهت ارتباط با کاربر به دو دسته تقسیم می شوند

1- سیستم های محاوره ای(Interactive): سیستم هائی هستند که در آنها کاربر به طور مستقیم و روی خط(on-line) با کامپیوتر در ارتباط است.کاربر دستوراتی را وارد می کند و منتظر پاسخ می ماند،پس از دریافت پاسخ مجددا دستوراتی را وارد می کند.

2- سیستم های دسته ای (batch) :سیستم هائی هستند که در آنها دریافت دستورات(برنامه های کاربر) و سپس اجرای آنها در دو مرحله انجام می گیرد.ابتدا برنامه هائی که عموما دارای نیازهای مشابه نظیر کامپایلر یکسان هستند در یک گروه به سیستم وارد شده و پس از بار شدن کامپایلر مورد نیازشان اجرای آنها به طور متوالی انجام میشود.

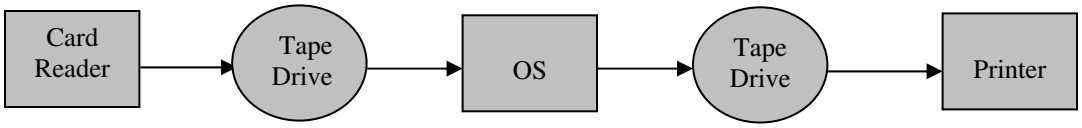
تقسیم بندی سیستم ها از جهت ارتباط با وسایل جانبی

1- سیستم های On-Line با ارتباط مستقیم:در این سیستمها پردازنده مستقیما با دستگاههای ورودی خروجی در **ارتباط** است.و به دلیل کند بودن این دستگاهها، کارائی پردازنده به مقدار قابل توجهی کاهش می یابد



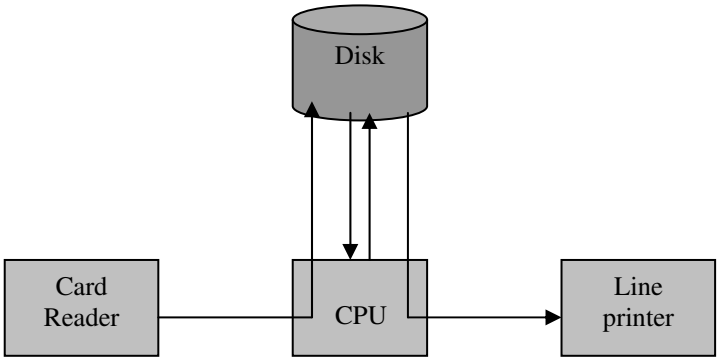
2- سیستم های Off-Line با ارتباط غیر مستقیم

در این سیستم ها پردازنده به طور مستقیم با دستگاههای ورودی و خروجی کند در ارتباط نیست.ابتدا عمل خواندن ورودی توسط کارت خوانهای مستقل از سیستم اصلی انجام میگیرد و ورودی به حافظه ی جانبی (نوار مغناطیسی) که سریعتر هستند منتقل شده،سپس این نوارها در سیستم اصلی استفاده میشود ،ارسال خروجی نیز بر روی نوار مغناطیسی انجام میگیردو نهایتا در محل دیگری بر روی چاپگرها ارسال می شود،درچنین حالتی امکان همپوشانی عمل I/O و کار پردازنده وجود دارد



بافرینگ: ناحیه ای از حافظه است که جهت ایجاد هماهنگی بین وسایل I/O کند و پردازنده استفاده می شودبافرینگ اجازه همپوشانی I/Oیک کارو پردازش همان کار را فراهم می سازد.

اسپولینگ: اسپولینگ یک رسانه ی ذخیره سازی سریع مانند دیسک مغناطیسی جهت انجام اعمال I/O استفاده میکند طوری که اطلاعات از وسایل ورودی بر روی دیسک ذخیره شده و CPU با دیسک در تماس است،همین طور اطلاعاتی که به وسایل خروجی مانندچاپگر می بایست ارسال شود ،بر روی دیسک ذخیره شده و بعد به چاپگر ارسال می شود. اسپولینگ مانند نفی است که به دور قرقره تابیده میشود به این امید که بعدا استفاده شود.



فرق بافرینگ و اسپولینگ چیست؟بافزینگ امکان همزمانی پردازش و I/O یک کار را به کمک حافظه ی اصلی فراهم میکند، در حالی که Spoling امکان هم زمانی پردازش و I/O چند کار را به کمک حافظه ی جانبی انجام می دهد.

چند برنامه گی(multi programing): در چند برنامه گی اجرای یک کار تا زمان نیاز به ورودی یا خروجی ادامه می یابد،سپس ورودی یا خروجی آن شروع شده و پردازنده اجرای کار دیگری را شروع کرده یا ادامه می دهد.

اشتراک زمانی(Time sharing): اشتراک زمانی شکل خاصی از چند برنامه گی است که در آن تعویض یک کار بر اساس یک معیار زمانی و نه بر اساس زمان نیاز آن کار به ورودی یا خروجی صورت می گیرد.اجرای یک کار تا پایان بازه ی زمانی ادامه یافته سپس پردازنده اجرای برنامه ی دیگری را شروع میکند.و عمل Switch بین پردازه ها چنان سریع است که هر کاربر فکر میکند سیستم به تنهائی در اختیار اوست

برنامه: یک موجود غیر فعال است که شامل یک یا چند فایل می باشد که بر روی دیسک ذخیره شده اند.

پردازه(فراروند):یک موجود فعال (Active) است که دارای ساختار خاصی به نام PCB می باشد.

مراحل ایجاد پردازه:

1-ایجاد پردازه:انتخاب یکی از کارها (برنامه های) موجود بر روی دیسک جهت تبدیل شدن به پردازه که به این عمل زمانبندی بلند مدت میگویند که توسط زمانبند کار انجام میشود.

2-گذراندن چرخه حالت :ایجاد وتخصیص ساختارهای لازم برای پردازه(بلاک کنترلی پردازه PCB: Process Control Block)

PCB شامل اطلاعات زیر است

1- حالت پردازه

2- شمارنده پردازه

3- ثبات های پردازنده

4- اطلاعات زمانبندی پردازنده

5- اطلاعات مدیریت حافظه

6- اطلاعات حسابرسی پردازه(میزان استفاده از منابع سیستم)

7- اطلاعات وضعیت پردازه در رابطه با دستگاههای ورودی و خروجی و فایل ها

3- خاتمه پردازه:قرار گرفتن پردازه ایجاد شده در صف پردازه های آماده اجرا

یک پردازه می تواند در چند حالت اصلی قرار گیرد.

1-حالت آماده(Ready state) :حالتی است که پردازه همه منابع مورد نیازش را در اختیار دارد و فقط منتظر CPU میباشد.

2-حالت منتظر یا مسدود(Waiting-Blocked) :حالتی است که در آن پردازه منتظر به دست آوردن یک منبع از سیستم می باشد در این حالت اگر پردازه، پردازنده را هم بدست آورد،قابل اجرا نیست پردازه از حالت اجرا با نیاز به یک منبع در این حالت قرار میگیرد.

3-حالت اجرا(Running Excuting):در این حالت پردازه همه منابع و پردازنده را در اختیار دارد و در حال اجراست؛ پردازه با بدست آوردن پردازنده از حالت آماده در این حالت قرار می گیرد.

4-حالت معلق:حالتی است که اگر پردازه به مدت زیادی در حالت آماده، منتظر CPU باشد و CPU به آن توجهی نکند که در آن صورت پردازه موقتا بر روی دیسک انتقال می یابد.با توجه مجدد CPU به این پردازه، پردازه از حالت معلق به حالت آماده میرود.

5-حالت کامل:هنگامی که پردازه به اندازه کافی وقت پردازنده را به دست آورده و کارش خاتمه یافته است (ولی هنوز از لیست پردازه های سیستم خارج نشده است)

نکته:در مورد پردازه های فرزند و پدر،خاتمه یک پردازه پدر می تواند منوط به خاتمه کلیه پردازه های فرزند آن گردد.(ابتدا پردازه های فرزند ختم میشوند سپس پردازه پدر به پایا ن می رسد) که به آن ختم پی در پی (Cascaded Termination)گفته می شود

...خاتمه فرزند فرزند← خاتمه فرزند←خاتمه پدر پردازه

خاتمه پردازه پدر← خاتمه فرزند← خاتمه فرزند فرزند...

صف های سیستم:

صف کار(Job Queue):در این صف برنامه هائی که منتظر تبدیل شدن به پردازه هستند قرار میگیرند مدیریت این صف به عهده زمانبند کار و یا زمانبند بلند مدت است.

صف آماده (Ready Queue):پردازه هائی که منتظر CPU هستند قرار می گیرند و توسط زمانبند کوتاه مدت و یا زمانبند CPU مدیریت میشود.

صف انتظار(waiting Queue): پردازه هائی که منتظر وسایل جانبی (I/O) هستند قرار می گیرند.

انواع زمانبندی:

1- **زمانبندی کوتاه مدت(Short Term scheduler):**در مرحله ی اجرای پردازه ها، به محض نیاز به ورودی یا خروجی ودر سیستم های اشتراک زمانی، به محض پایان برش زمانی، ودر سیستم ها ئی که امکان ایجاد پردازنده فرزند وجود دارد، به محض یک درخواست برای

ایجاد فرزند(Fork) (در حالتی که پردازنده پدر می بایست متوقف شود) ،پردازه در حال اجرا باید از پردازنده و از صف پردازه هائی آماده اجرا خارج شود و بر اساس الگوریتم زمانبندی یک پردازه دیگر انتخاب شده و به پردازنده بار (Load) شود.این مر حله تعویض پردازه، عمل تعویض متن(contest switching) نام دارد و کل اعمال فوق توسط مدیر زمانبند کوتاه مدت انجام میشود این زمانبند با توجه به کوتاه بودن برش زمانی، در فواصل کم و با فرکانس بالا انجام میشود.تعویض متن توسط بخشی از سیستم عامل به نام Dispatcher انجام میشود.

2- زمانبندی میان مدت(Medium Term scheduler):در بعضی شرایط به دلیل زیاد شدن تعداد پردازه های موجود در چرخه حالت پردازه ها(چرخه آماده–اجرا –منتظر) و در نتیجه کم شدن حافظه ی آزاد سیستم و کاهش کارائی، مناسب است تعدادی از پردازه ها از حالت فعال در پردازنده خارج شده (به طور موقت) و پس ازاتمام انجام اجرای تعدادی از پردازه ها، اجرای آنها ادامه یابد . عمل انتقال پردازه های موجود در حافظه ی اصلی ودر حالت آماده به حافظه ی جانبی به منظور کاهش بار سیستم **Swap out** و انتقال دوباره پردازه های آماده از حافظه ی جانبی به حافظه ی اصلی و فعال شدن دوباره آنها، **Swap in** نام دارد که هر دو توسط بخشی از سیستم عامل به نام مدیریت زمانبند میان مدت انجام میشود.

3–زمانبند بلند مدت:انتخاب یکی از کارهای موجود در سیستم جهت تبدیل شدن به پردازه، چون این عمل در فواصل طولانی انجام می شود به زمانبندی بلند مدت مشهور است.

پایان جلسه ی اول

جلسه ی دوم:

عموما کارها در دو دسته قرار می گیرند

1- **کارهای I/O bound (I/O Limited):** کارهائی که بخش زیادی از اجرای آنها در ارتباط با دستگاههای ورودی/خروجی بوده و محاسبات زیادی ندارند.مثل برنامه ا ی که می بایست کارنامه ی دانشجویان را چاپ کند

2- **کارهای CPU bound (CPU Limited):** کار هائی که حجم زیادی محاسبات داشته و بخش عمده نیاز آنها برای اجرا وقت پردازنده است.مثل برنامه ای که می بایست یک دستگاه معادله ی صد مجهولی را حل کند

نکته:

در مورد کاهای CPU bound و I/O bound استفاده از باغر تاثیر زیادی در بهبود اجرا ندارد.

زمانبند بلند مدت می بایست تلفیق مناسبی از کارهای CPU bound و I/O bound را انتخاب کند.

حالت پایدار: زمانی که نرخ ایجاد پردازها برابر با نرخ خروج یا خاتمه پردازه ها باشد .بنا بر این زمانبند بلند مدت سعی می کند که سیستم به حالت پایدار برسد.

الگوریتم های زمانبندی کوتاه مدت(پردازنده):

عمل زمانبندی کوتاه مدت بر اساس الگوریتم های زمانبندی انجام می شود.این اگوریتم ها سعی در برآورده کردن معیارهای زیر را دارند.

1- **عدالت:** یعنی سهم پردازه ها از CPU به به طور منصفانه باشد

2- **افزایش کارائی:** هدف از کارائی مشغول بودن CPU به طور ایده آل است.

3- **افزایش گذردهی:** منظور تعداد پردازه های انجام شده در واحد معینی از زمان می باشد.

4- **کاهش زمان پاسخ:** زمان ما بین مطرح شدن یک پردازه تا اولین اجرای آن پردازه می باشد.

این اهداف کاملا باهم در تناقض اند بنا براین می بایست سعی شود مصالحه ای بین این اهداف صورت گیرد.

- **زمان انتظار(waiting Time):** مدت زمانی را که پردازه در حالت آماده است و منتظر CPU می باشد را زمان انتظار گویند.
- **زمان اجرا(Runing Time):** مدت زمانی را که پردازه بر روی CPU در حال اجراست را زمان اجرا گویند
- **زمان I/O:** مدت زمانی است که پردازه نیاز به عمل I/O دارد
- **زمان گردش کار(Turnatound Time):** مدت زمان ما بین ورود یک پردازه به سیستم تا اجرای کامل یک پردازه را زمان گردش کار گویند.
- **زمان تعویض متن+زمان I/O + زمان اجرا +زمان انتظار = زمان گردش کار**
- **زمان تعویض متن:** زمانی است که برای تعویض پردازه ها بر روی CPU صرف میشود.که این عمل شامل ذخیره حالت پردازه در حال اجراو Update ثبات های سیستم به مقادیر پردازه جدید می باشد.

الکوریتم های زمانبندی به دو دسته تقسیم می شوند.

1-انحصاری یا انقطاع نا پذیر(Preemptive): در این اگورریتم ها تا زمان خاتمه پردازه و یا تا زمان نیاز به عمل I/O نمی توان CPU را از پردازه در حال اجرا گرفت و به پردازه دیگری انتساب داد.

2-غیر انحصاری یا انقطاع پذیر(Non Preemptive): در این الگوریتم ها در پایان برش زمانی (Time Slice) و یا تغییر شرایط سیستم، مدیر زمانبندی می تواند کنترل پردازنده را از یک پردازه در حال اجرا گرفته و به پردازه دیگری بدهد.

الگوریتم الویت با اولین ورودی-FCFS-(Frist Come Frist Service یا Frist In Frist Out)

این الگوریتم به سادگی هر پردازه ورودی را در صف FIFO قرار داده و از سر صف اجرایش را شروع می کند و اجرای پردازه ها را به طور انحصاری انجام میدهد.

مزایا:

سادگی اجرا- عملی بودن

معایب:

1-زیاد بودن میانگین زمان انتظار(زمان گردش کار)

2-انقطاع نا پذیری و غیر قابل استفاده بودن در سیستم های اشتراک زمانی است

الگوریتم الویت با کوتاهترین کار-SJF (Shortest Job Frist):

این الگوریتم از بین پردازه های موجود در صف آماده، پردازه ای را جهت اجرا انتخاب می کند که زمان اجرای کمتری نیاز داشته باشدو پردازه در حال اجرا اجرایش را به صورت انقطاع ناپذیر ادامه می دهد.

معایب:

1- نیاز به داشتن اطلاعات مقاطع زمانی مورد نیاز قبل از شروع اجرا

2- احتمال به تعویق افتادن کارهای طولانی

3- انقطاع نا پذیری

مزایا:دارا بودن کمترین زمان برگشت یا زمان انتظار ما بین تمام الگوریتم ها

اگوریتم الویت با کمترین زمان باقیمانده-SRT (Shortest Remaining Time)

در این الگوریتم انتخاب بر اساس کمترین زمان مورد نیاز برای کامل شدن صورت میگیرد. این الگوریتم غیر انحصاری بوده و با ورود هر پردازه به صف آماده، بر رسی زمان باقیمانده پردازه ها انجام می گیرد، اگر پردازه تازه وارد شده، زمان کمتری برای کامل شدن لازم دارد، پردازنده در اختیار آن قرار می گیرد.

معایب:احتمال تعویق کارهای طولانی-نیاز به دانستن زمان مورد نیاز پردازه ها

مزایا:انقطاع پذیری- زمان پاسخ نسبتا مناسب

مثال:در سیستمی سه پردازه P_0 و P_1 و P_2 مطابق جدول زیر با زمان های ورود و اجرای داده شده قرار دارند مطلوب است محاسبه میانگین زمان انتظار و میانگین زمان گردش کار در حالت های زیر

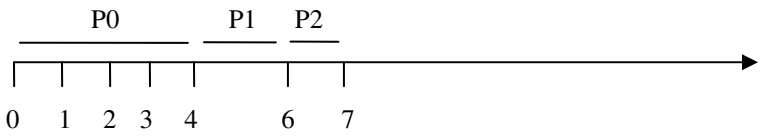
پردازه	زمان رسیدن	زمان اجرا
P_0	t	4
P_1	t+1	2
P_2	t+2	1

الف:برای زمانبندی پردازه ها از الگوریتم FCFS استفاده شود.

ب:برای زمانبندی پردازه ها از الگوریتم SJF استفاده شود.

ج:برای زمانبندی پردازه ها از الگوریتم SRT استفاده شود.

الف:



$$\text{میانگین زمان انتظار} = \frac{\text{مجموع زمان انتظار تمام پردازه ها}}{\text{تعداد پردازه ها}}$$

$$\text{میانگین زمان گردش کار} = \frac{\text{زمان اجرای پردازه+زمان انتظار}}{\text{تعداد پردازه ها}}$$

$$\text{میانگین زمان گردش کار} = \frac{(0+4)+(3+2)+(4+1)}{3} = \frac{14}{3}$$

$$\text{میانگین زمان انتظار} = \frac{0+3+4}{3} = \frac{7}{3}$$

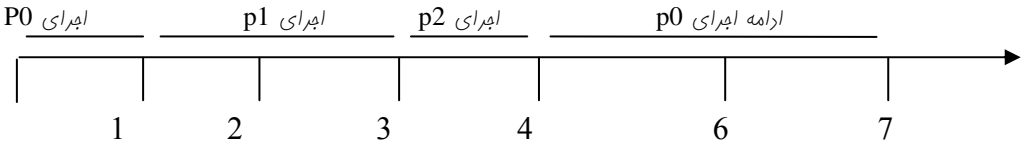
ب)در این شرایط اگر پردازه ای شروع به اجرا شود تا انتها ادامه می یابد در مرحله ی بعد پردازه ای برای اجرا انتخاب می شود که کمترین زمان اجرا را دارد بنا براین اول P0 (در این لحظه فقط همین را داریم) تا لحظه ی4 اجرا میشود در این لحظه دو تا پردازه آماده داریم که P2 زمان کمتری را برای اجرا می خواهد پس تا لحظه ی 5 اجرا می شود از 5 تا 7 زمان اجرای P1 می باشد.

زمان انتظار : P0=0s , P1=4s , P2=2s

$$\text{میانگین زمان انتظار} = \frac{0+4+2}{3} = 2$$

$$\text{میانگین زمان گردش کار} = \frac{(0+4)+(4+2)+(2+1)}{3} = \frac{13}{3}$$

ج) در این شرایط در تمامی لحظه های اجرا اگر پردازه ای از را ه برسد که زمان اجرایش کمتر از زمان باقیمانده پردازه در حال اجرا باشد پردازنده را در اختیار گرفته و شروع به اجرا می کند زمان انتظار : P0=3s ,p1=0s,p2=1S



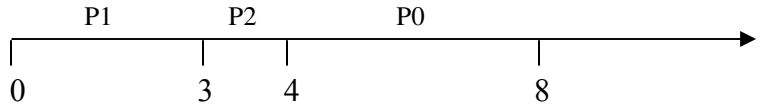
$$\text{میانگین زمان گردش کار} = \frac{(3+4)+(0+2)+(1+1)}{3} = \frac{11}{3}$$

$$\text{میانگین زمان انتظار} = \frac{3+0+1}{3} = \frac{4}{3}$$

مثال: مثال قبل را با جدول زیر حل کنید(برای قسمت ب و ج)

پردازه	زمان رسیدن	زمان اجرا
P_0	0	4
P_1	0	3
P_2	1	1

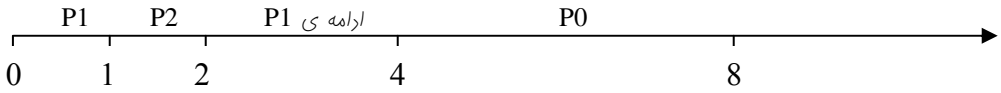
ب) SJF



میانگین زمان انتظار $AWT = \frac{4+0+2}{3} = 2$

میانگین زمان گردش $ATT = \frac{(4+4)+(0+3)+(2+1)}{3} = \frac{14}{3}$

ج) SRT



$AWT = \frac{4+1+0}{3} = \frac{5}{3}$

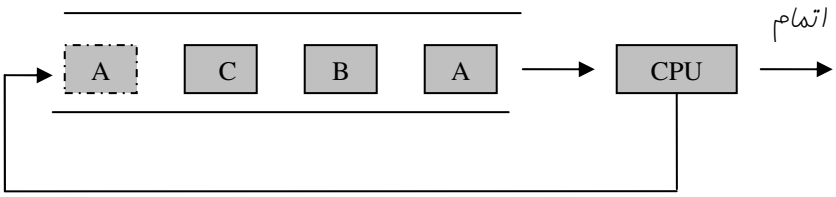
$ATT = \frac{(4+4)+(1+3)+(0+1)}{3} = \frac{13}{3}$

پایان جلسه دوم

جلسه ی سوم:

الگوریتم نوبتی (Round Robin: RR) : در این الگوریتم با در نظر گرفتن یک برهه زمانی (time slice) یا کوانتوم زمانی پردازه های موجود در صف آماده هر کدام به اندازه این برهه زمانی CPU را بدست آورده به طوری که اگر در برهه زمانی شان اجرای خود را به اتمام نرسانند مجددا به انتهای صف منتقل می شوند.

- این الگوریتم بیشتر در روش های اشتراک زمانی (Time sharing) استفاده می شود.



انتظار برای I/O یا پایان یافتن برش زمانی

مزایا:

1- زمان پاسخ نسبتا مناسب (نوبه اجرا به همه می رسد)
2- غیر انحصاری بودن
3- عدم نیاز به دانستن مقاطع زمانی پردازه ها

معایب:

1- نیاز به دقت در تنظیم برش زمانی
2- کاهش کارائی پردازنده به خاطر زمان های تعویض متن در برهه های زمانی خیلی کوتاه
3- شباهت به الگوریتم FCFS برای برش های زمانی طولانی

☐ کوانتوم زمانی بین 10 تا 100 میلی ثانیه است .

مثال: سه پردازه زیر را در نظر بگیرید ، اگر از کوانتوم زمانی 4 میلی ثانیه استفاده شود ، میانگین زمان انتظار و میانگین زمان گردش کار را بیابید.(هر سه در لحظه صفر واردشده اند ولی الویت به ترتیب با شماره است)

پردازه	زمان اجرا
p_1	24
p_2	3
p_3	3

نمودارگانت پردازش ها به صورت زیر خواهد بود.

P1	P2	P3	P1	P1	P1	P1	P1	

0 4 7 10 14 18 22 26 30

میانگین زمان انتظار $= \frac{6+4+7}{3} = \frac{17}{3} ms$

میانگین زمان گردش کار $= \frac{(6+24)+(4+3)+(7+3)}{3} = \frac{47}{3} ms$

مثال:

چهار پردازنده مطابق جدول زیر در سیستم وجود دارند. اگر از روش RR با برش زمانی 1ms استفاده شود و از سربار ناشی از تعویض متن فرایند ها صرف نظر شود میانگین زمان انتظار پردازه ها چقدر است (فرض شود پردازه ای که از راه می رسد به ابتدای صف منتقل می شود)

پردازش ها	زمان ورود	زمان پردازش
p_1	0	3
p_2	1	5
p_3	3	2
p_4	9	2

P1	P2	P1	P3	P2	P1	P3	P2	P2	P4	P2	P4

0 1 2 3 4 5 6 7 8 9 10 11 12

متوسط زمان انتظار $= \frac{3+5+2+1}{4} = \frac{11}{4} ms$

الگوریتم زمانبندی الویت دار

الویت با بالاترین نسبت پاسخ (Highest Respones Ratio Next -HRN)

$$\text{الویت} = \frac{\text{زمان سرویس + زمان انتظار}}{\text{زمان سرویس}}$$

الگوریتم های الویت دار با در نظر گرفتن یک الویت برای هر کدام از پردازه های موجود در سیستم ، پردازه ای که بالاترین الویت را داشته باشد اجرایش را به صورت انحصاری انجام می دهد ، الویت هائی که می توان در این الگوریتم در نظر گرفت عبارتند از

1- نسبت مقاطع زمانی ورودی یا خروجی به مقاطع زمانی پردازنده (I/O Burst به CPU Burst)
2- محدودیت زمانی ، نیاز های حافظه
3- تعداد فایل های باز شده
4- اهمیت پردازه و بخش ارائه دهنده کار
5- الویت نسبت پاسخ (فرمول بالائی)

مزایا:

1- طبق این الگوریتم پردازه های کوتاه در ابتدا الویت بالا پیدا می کنند ، چون زمان سرویس(زمان مورد نیاز جهت اجرا) کمتری نیاز دارند

2- پردازه هائی که مدت زیادی منتظر مانده اندنیز الویت بالا پیدا می کنند ، پس این شانس را پیدا می کنند که اجرا شوند.

معایب:

1- می بایست زمان سرویس را از قبل معلوم کرد
2- انحصاری است

مثال:

پردازه های زیر را در نظر بگیرید که همگی در لحظه ی صفر به ترتیب داده شده رسیده اند میانگین زمان انتظار را بیابید و جهت زمانبندی پردازه ها از الگوریتم الویت دار استفاده کنید و پردازه ای که عدد الویت آن کمتر است ، الویت بالاتری دارد.

حل:نمودار گانت به صورت زیر خواهدبود

P2	P5	P1	P3	P4

0 1 6 16 18 19

پردازه ها	زمان اجرا	الویت
p_1	10	3
p_2	1	1
p_3	2	3
p_4	1	4
p_5	5	2

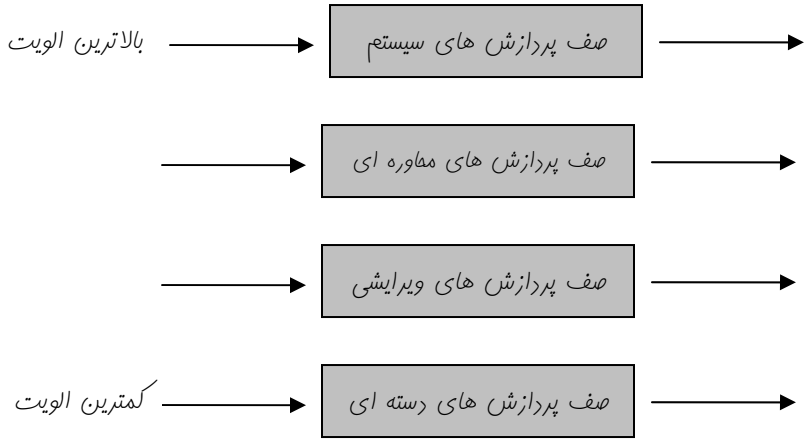
$$\text{میانگین زمان انتظار} = \frac{6+0+16+18+1}{5} = \frac{41}{5} ms$$

الگوریتم صف چند سطحی(Multi Level Queue:MLQ):

در الگوریتم صف چند سطحی پردازه ها در صف های مختلف ، که هر صف الویت خاصی دارد قرار می گیرند و در صف های مختلف از الگوریتم های زمانبندی مختلفی استفاده می شود ، پارامتر هائی که در این الگوریتم می بایست مشخص نمود عبارتند از :

1- تعداد صف ها
2- الگوریتم زمانبندی استفاده شده در هر صف
3- الویت صف ها نسبت به هم

- اگر در صفی با الویت بالا پردازه ای وجود داشته باشد CPU در ابتدا پردازه های آن صف را سرویس داده و در صورتی که صف های الویت بالاتر خالی شود به سراغ صف الویت پایین می رود.
- به عنوان مثال سیستمی می تواند 4 صف آماده با الویت های زیر داشته باشد



الگوریتم صف باز خورد چند سطحی(Multi Level Feedback Queue : MFQ)

این الگوریتم مانند الگوریتم MLQ می باشد با این تفاوت که در این الگوریتم امکان حرکت پردازه ها بین صف های مختلف نیز وجود دارد ، در این الگوریتم علاوه بر مشخص نمودن پارامتر های MLQ موارد زیر نیز می بایست مشخص شود .

1- چه موقع یک پردازه از صف بالا به صف پایین مهاجرت می کند
2- چه موقع یک پردازه از صف پایین به صف بالا مهاجرت می کند

مثال:

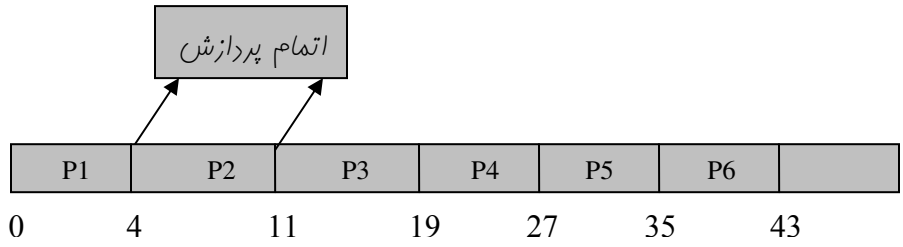
یک سیستم با الگوریتم MFQ یا (MLF) سه سطحی را در نظر بگیرید . اگر سطح اول از الگوریتم RR با کوانتوم زمانی 8 میکرو ثانیه و صف دوم از الگوریتم RR با کوانتوم زمانی 16 میکرو ثانیه و صف سوم با الگوریتم FCFS زمانبندی شوند . اگر 6 پردازه همگی در زمان صفر با زمانهای اجرای 4 ، 7 ، 12 ، 20 ، 25 و 30 میکرو ثانیه وارد سیستم می شوندمیانگین زمان برگشت و میانگین زمان انتظار

را در این سیستم بیابید(هر پردازه به محض پایان کوانتوم زمانی از صف بالاتر به صف پایین تر مهاجرت می کند)(کارشناسی ارشد- دولتی 80)

حل:

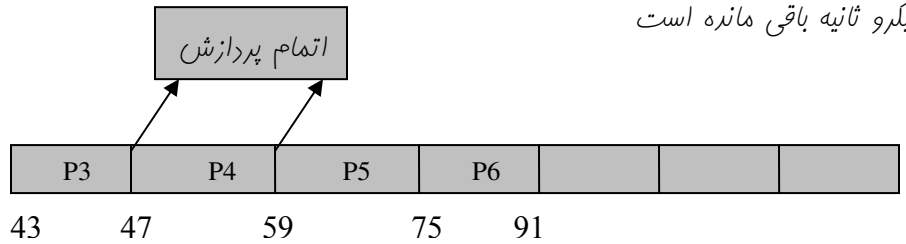
ابتدا برنامه ها در صف اول با TS=8 μ S قرار می گیرند و اگر در مدت 8 میکرو ثانیه تمام نشوند به صف دوم با TS=16 μ S منتقل می گردند باز هم اگر در این مدت 16 میکرو ثانیه تمام نشوند وارد صف سوم با الگوریتم FCFS می گردند . بنابراین نمودار زمانی این پردازش ها به صورت شکل زیر خواهد بود

صف اول با TS=8 μ S



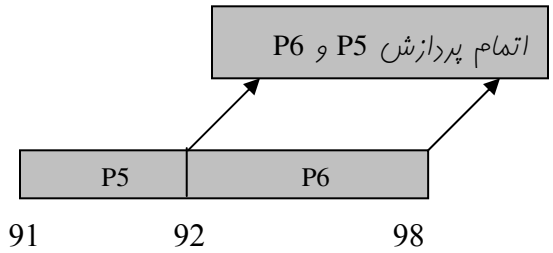
پس از یکبار اجرای چرخشی P1 و P2 تمام می شوند در حالی که از برنامه های P3 ، P4 ، P5 و P6 به ترتیب زمان های 4 ، 12 ، 17 و 22 میکرو ثانیه باقی مانده است

صف دوم با TS=16 μ s



در انتهای مرحله ی دوم P3 و P4 تمام شده و از برنامه P5 به اندازه 1 و از برنامه P6 به اندازه 6 میکرو ثانیه باقی مانده است ، این دو پردازش در مرحله ی آخر به صورت FCFS زمانبندی می شوند.

صف سوم FCFS:



میانگین زمان انتظار=$\frac{0+4+35+39+67+68}{6}=35/5$

میانگین زمان برگشت = $\frac{(0+4)+(4+7)+(35+12)+(39+20)+(67+25)+(68+30)}{6}=51/83$

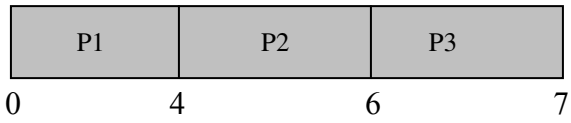
مثال:سه پردازه دسته ای (Batch) P_1 ، P_2 ، P_3 با زمان اجرا و زمان وارد شدن زیر را در نظر بگیرید متوسط زمان پاسخگوئی را با تمامی الگوریتم ها به جز الگوریتم های MFQ و MLQ بیابید.(ارشد- 83)

p_3	1	t+3	1

	الویت	زمان وارد شدن	زمان اجرا
p_1	2	t	4
p_2	0	t	2

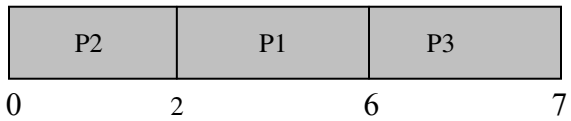
حل: t را صفر در نظر می گیریم

الف) FCFS



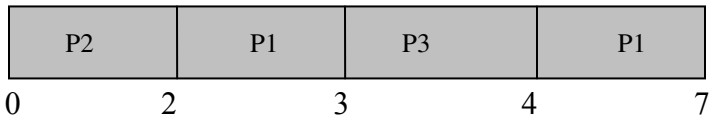
$$AWT=\frac{(0+4)+(4+2)+(3+1)}{3}=\frac{14}{3}$$

ب)SJF



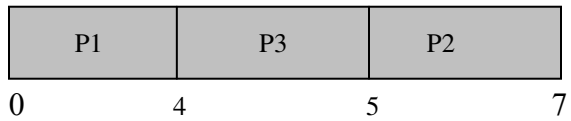
$$AWT=\frac{(2+4)+(0+2)+(3+1)}{3}=\frac{12}{3}$$

ج) SRT



$$AWT=\frac{(3+4)+(0+2)+(0+1)}{3}=\frac{10}{3}$$

د) الگوریتم نوبتی(Priority):



$$AWT=\frac{(0+4)+(5+2)+(1+1)}{3}=\frac{13}{3}$$

مثال:

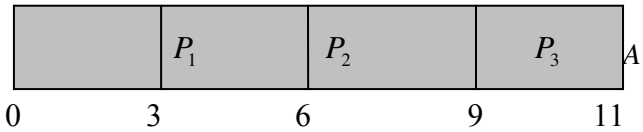
چهار پردازه P_0 ، P_1 ، P_2 ، P_3 با مشخصات زیر در نظر بگیرید .میانگین زمان پاسخ دهی آنها را در الگوریتم های زمانبندی زیر بیابید .(فرض کنید تکه زمانی معادل یک واحد زمانی است . در مورد سیاست با گردش نوبت پردازه ای که وارد سیستم می شود در همان ابتدای ورودش اجرای آن آغاز می شود)

الف) FCFS

ب) RR

نام پردازه	زمان ورود به سیستم	زمان پردازش
p_0	0	3
p_1	1	3
p_2	4	3
p_3	6	2

حل: الف) FCFS



$$AWT=\frac{(0+3)+(2+3)+(2+3)+(3+2)}{4}=\frac{18}{4}$$

ب) RR

P_0	P_1	P_0	P_1	P_2	P_0	P_3	P_1	P_2	P_3	P_2

0 1 2 3 4 5 6 7 8 9 10 11

$$AWT = \frac{(3+9)+(4+3)+(4+3)+(2+2)}{4} = \frac{24}{4}$$

جلسه چهارم

رابطه بین پردازه ها(Interprocess Communication)

پردازه ها از جهت وابستگی به یکدیگر در دو دسته قرار می گیرند و توجه به این مسئله در نحوه خاتمه آنها حائز اهمیت است

الف- پردازه های مستقل: هر پردازه ای که داده ای را (چه موقت و چه دائمی) با دیگر پردازه ها به اشتراک نگذارد مستقل است و یا به عبارت دیگر یک پردازه در صورتی مستقل از دیگر پردازه ها است که شرایط زیر را برآورده کند.

1- حالت آن به اشتراک گذارده نشده باشد.

2- نتیجه اجرای آن کاملا مشخص باشد.

3- نتیجه اجرا در دفعات مختلف به ازای ورودی های یکسان، مساوی باشد.

4- توقف و شروع آن بدون تاثیر بر سایر پردازه ها امکان پذیر باشد.

ب- پردازه های همکاری کننده: اگر دو پردازه نسبت به هم یکی از شرایط فوق را نداشته باشد گوئیم پردازه ها نسبت به هم همکارند به عبارت دیگر پردازه هائی هستند که شرایط زیر در مورد آنها صادق باشد.

1- حالت آن به اشتراک گذارده شود

2- نتیجه اجرا به دلیل وابستگی به سایر پردازه ها کاملا مشخص نباشد

3- نتیجه اجرا برای ورود ی های یکسان در دفعات مختلف، یکسا ن نباشد

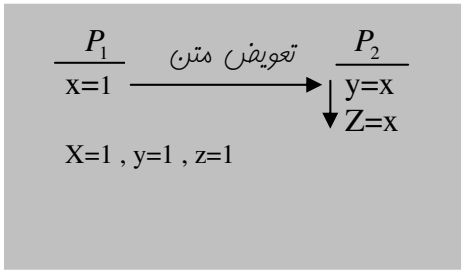
منظور از اجرای Interleaved چیست؟

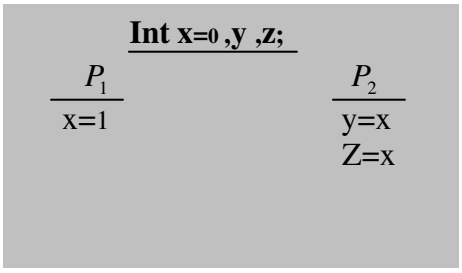
به این معناست که در بین اجرا ی یک پردازه در هر زمان امکان وقفه یا تعویض متن (Context Switching) به پردازه دیگر وجود دارد

مثال: اگر دو پردازه P_1 و P_2 به صورت همروند اجرا شوندمقادیر نهائی x ، y و z را بیابید

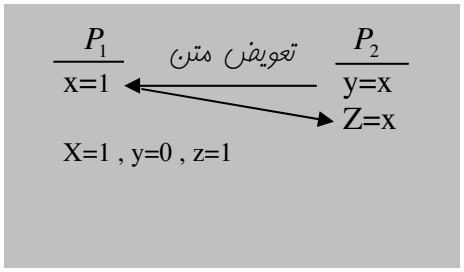
حل:در اجرای همروند این دو پردازه سه ترتیب زیر امکان پذیر است

الف-

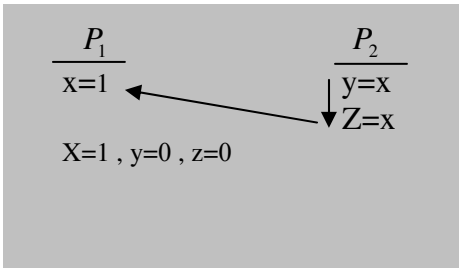




ب-



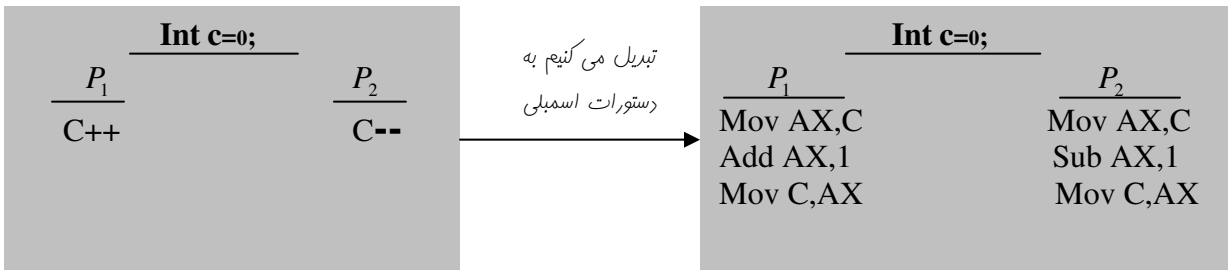
ج-



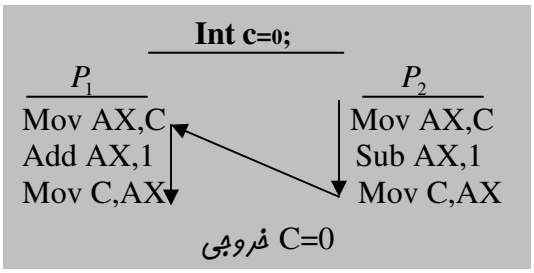
منظور از همگام سازی پردازه ها(Synchronization) چیست؟

زمانی که پردازه ها به هم وابسته اند نتیجه اجرا به ترتیب اجرای دستورات برنامه بستگی دارد، بنا براین می بایست با توجه به نتایج مورد انتظار ترتیب اجرای دستورات پردازه ها را مشخص کرد که به این مسئله همگام سازی پردازه ها گویند

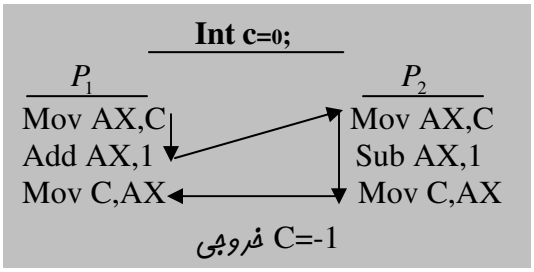
مثال: اگر دو پردازه P_1 و P_2 به صورت همروند اجرا شوندمقادیر نهائی c بیابید.



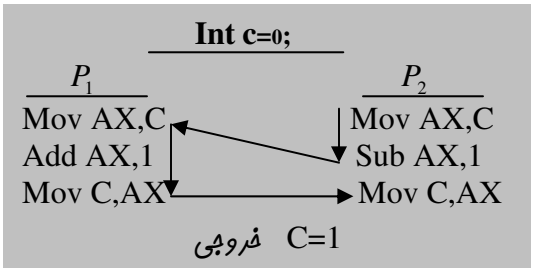
الف-



ب-



ج-



به علت اینکه هر دو پردازه از متغیر مشترک C استفاده کرده اند، مقدار نهائی C وابسته به ترتیب اجرای پردازه ها می باشد که در این مثال بسته به ترتیب اجرای دستورات C می تواند مقادیر 0,-1,1 را اتخاذ نماید، ولی نتیجه مورد نظر صفر می باشد.

منظور از وضیعت مسابقه یا Race Condition چیست؟

وضعیت هائی که در آن فرایند های متعددی، داده یکسانی را به طور همروند دستیابی و دستکاری می کنند و حاصل اجرا بستگی به ترتیب خاص دسترسی ها دارد ، وضیعت مسابقه یا Race Condition گفته می شود.

نواحی بحرانی(Critical Section):

برای جلوگیری از شرایط رقابتی باید راهی را پیدا کنیم که از خواندن و نوشتن داده های مشترک ،به طور همزمان، توسط بیش از یک پروسس جلوگیری به عمل آید. به عبارت دیگر ما به« انحصار متقابل» یا Mutual Eclusion نیاز داریم، به عبارتی دیگر اگر یکی از پردازه ها در حال استفاده از داده مشترک است باید مطمئن باشیم که دیگر پردازه ها، در آن زمان از انجام همان کار محروم می باشند.

- بخشی از برنامه که به حافظه اشتراکی دسترسی دارد را قسمت یا ناحیه بحرانی (Critical Section) می نامیم.
- هر پردازه برای ورود به بخش بحرانی اش باید اجازه بگیرد. بخشی از کد پردازه که این اجازه گرفتن را پیاده سازی می کند بخش ورودی یا Entry section نام دارد.

بخش بحرانی می تواند با بخش خروجی یا exit section دنبال شود . این بخش خروجی کاری می کند که پردازه های دیگر بتوانندوارد ناحیه بحرانی شان بشوند. بقیه کد پردازش را بخش باقی مانده یا remainder section گوئیم. بنا براین ساختار کلی پردازش ها به صورت زیر می باشد.

```
While(True){
        Entry section
        Critical_section();
        exit section
        Remainder_section();
}
```

باید جهت رفع مشکل وضیعت مسابقه چهار شرط زیر رعایت گردد تا یک راه حل خوب بدست آید

1-شرط انحصار متقابل(مانعه الجمعی Mutual Exclusion):

هنگامی که پردازشی در ناحیه بحرانی اش اجرا می گردد، هیچ پردازش دیگری نباید در ناحیه بحرانی حضور داشته باشد.

2- شرط پیشرفت یا پیشروی(Progress):

هنگامی که هیچ پردازشی در قسمت بحرانی در حال اجرا نباشد و تقاضاهائی برای ورود به بخش بحرانی وجود دارد، فقط پردازش هائی در تصمیم گیری برای ورود دخالت می کنند که هنوز به ناحیه بحرانی شان نرسیده باشند. به عبارت دیگر اگر پردازشی در قسمت باقی مانده(remainder) خود باشد، در تصمیم گیری اینکه که کدام پردازه وارد بخش بحرانی شود، شرکت داده نمی شود.به عبارت دیگر هیچ پردازشی نباید از بیرون ناحیه بحرانی خود امکان بلوکه کردن پردازش های دیگر را داشته باشد.

3- شرط انتظار مقید یا محدود(Bounded Waiting):یک برنامه منتظر ورود به ناحیه بحرانی،نباید به طور نامحدود در حالت انتظار باقی بماند

4- هر پردازشی با سرعت غیر صفر اجرا می شود ولی هیچ فرضی در مورد سرعت نسبی n پردازش و نیز تعداد CPU ها نمی کنیم

روش های جلوگیری از مسابقه(همزمانی پردازش ها)

1-غیر فعال ساختن وقفه ها:

هر پردازه بلا فاصله پس ا ز ورود به ناحیه بحرانی اش کلیه وقفه ها را از کار بیندازد و درست قبل از خروج از ناحیه بحرانی دوباره همه آنها را فعال کند. با خاموش ساختن وقفه ها CPU به هیچ عنوان نمی تواند از پردازشی به پردازش دیگر سوئیچ کند. پس پردازه می تواند بدون ترس از مداخله دیگر پردازه ها به دستکاری قسمت مشترک بپردازد .

این روش دو مشکل دارد

1-ممکن است کاربر وقفه ها را خاموش کند ولی دوباره آنهارا فعال نسازد(یادش برود) بدین ترتیب سیستم از کار خواهد افتاد. پس اعطای قدرت غیر فعال ساختن وقفه ها به پردازش کاربران عاقلانه نیست.

2-در سیستم های چند پردازنده ای غیر فعال ساختن وقفه ها، فقط در CPU ای اثر دارد که دستور از کار انداختن وقفه را اجرا می کند، بقیه CPU ها می توانند کار خودشان را ادامه داده و به حافظه مشترک دستیابی پیدا کنند.

2-استفاده از متغییر های قفل(Lock Variables):

فرض کنید یک متغیر قفل یکتا ومشترک با مقدار اولیه صفر وجود دارد(متغیر مثلا با نام Lock). هنگامی که پردازشی می خواهد وارد ناحیه بحرانی خود شود، ابتدا Lock را آزمایش می کند اگر Lock=0 بود آن را برابر "1" کرده و وارد ناحیه بحرانی می شود ولی اگر Lock=1 بود، باید در یک حلقه منتظر بماند تا Lock برابر صفر شود. بنا براین "0" به این معناست که هیچ پردازشی در ناحیه بحرانی نیست و "1" به این معناست که پردازشی در ناحیه بحرانی اش قرار دارد. کد زیر این روش را نشان می دهد، مقدار اولیه Lock برابرصفر است.

نکته:این روش شرط اصلی انحصار متقابل را ندارد

	P_0	P_1
قسمت ورودی	While(Lock==1); Lock=1;	While(Lock==1); Lock=1;
	Critical_Section();	Critical_Section();
قسمت خروجی	Lock=0;	Lock=0;

توضیح:

لحظه ای را تصور کنید که متغیر Lock برابر صفر است، پردازش P_0 در حلقه While متغیر Lock را چک می کند و چون برابر صفر است به سراغ خط بعدی می رود تا Lock را برابر "1" کند . ولی قبل از آنکه عدد "1" را در Lock بریزد، CPU به پردازش P_1 سوئیچ می کند. در این حال P_1 نیز متغیر Lock را برابر صفر می بیند و از حلقه While خارج می شود، آنگاه در ادامه Lock را برابر "1" کرده و وارد ناحیه بحرانی خود می شود . حال دوباره پردازنده به P_0 سوئیچ می کند، عدد "1" را در Lock ریخته و وارد ناحیه بحرانی P_0 می شود. یعنی هر دو پردازش P_0 و P_1 همزمان در ناحیه بحرانی می باشند!

3-روش تناوب قطعی (Strict Alternation):

در این روش از یک متغیر نوبت استفاده می شود طوریکه اگر دو پردازه P_1 و P_2 داشته باشیم بعد از این که یکی از پردازه ها وارد ناحیه بحرانی شد، تا این که پردازه دیگر وارد ناحیه بحرانی نشود، این پردازه نمی تواند مجددا وارد ناحیه بحرانی بشود .

```
P_i
Other = 1-i
While(Turn==Other);
Critical_Section();
Turn=Other;
Remainder Code
```

نکته: با این که این الگوریتم شرط انحصار متقابل را داردولی شرط پیشرفت در آن برقرار نیست

P_0	P_1
While(Turn !=0);	While(turn !=1);
Critical_Section();	Critical_Section();
Turn=1;	Turn=0;
Remainder	Remainder

توضیح: حالتی را در نظر بگیرید که پردازه P_0 وارد ناحیه بحرانی می شود که هنگام خروج از ناحیه بحرانی مقدار Turn را برابر "1 "قرار می دهد و در این موقع پردازه P_1 می تواند وارد ناحیه بحرانی شود، پردازه P_1 پس از خروج از ناحیه بحرانی Turn را برابر صفر قرار می دهد،فرض کنید پردازه P_1 در Remainder Code باشد اگر در همین اثنا پردازه P_0 بخواهد وارد ناحیه بحرانی بشود چون Turn برابر صفر می باشد، وارد ناحیه بحرانی می شود این پردازه پس از خروج از ناحیه بحرانی Turn را برابر "1 " می کند . حال اگر قسمت Remainder Code مربوط به P_1 طولانی باشد، اگر P_0 بخواهد مجددا وارد ناحیه بحرانی بشود، نمی تواند زیرا Turn برابر با "1 " است، پس پردازه P_1 که در قسمت Remainder Code خود قرار دارد در تصمیم گیری ورود به ناحیه بحرانی دخالت دارد و این یعنی نقض شرط پیشرفت

4- استفاده از دستور سخت افزاری(TSL):

دستورات اتمیک: دستوراتی هستند که تجزیه نا پذیرند(وقفه ناپذیر)

Mov Ax, Count اتمیک است

Count++ اتمیک نیست

در این روش از دستور اتمیک TSL استفاده می شود که طرز کار این دستور به صورت زیر است

پردازه ها در هنگام ورود به ناحیه بحرانی دستور TSL را فراخوانی می کنند، که اگر مقدار Lock مخالف صفر باشد در حلقه گیر کرده و منتظر می مانند و اگر مقدار Lock صفر باشد، پردازه ای که این دستور را اجرا کرده مقدار Lock را صفر می بیند و وارد ناحیه بحرانی می شود(در این لحظه مقدار Lock برابر "1 " می شود)، اگر در همین حین پردازه ی دیگری TSL را اجرا کند ، چون مقدار Lock برابر "1 " است نمی تواند وارد ناحیه بحرانی بشود، پردازه ها پس از خروج از ناحیه بحرانی مقدار Lock را صفر می کنند تا دیگر پردازه ها بتوانند وارد ناحیه بحرانی بشوند.(دقت شود که در اولین اجرا مقدار Lock برابر صفر است)

تابع Tsl را می توانیم به این شکل بنویسیم

P_i

```
While(1) {
While(TSL(& Lock))
    Critical Section
Lock=0 ;
    Rimainder Section
}
```

```
Tsl(x){
Int temp;
Temp=x;
x=1;
return (temp);
}
```

پایان جلسه چهارم

جلسه پنجم

5- راه حل پیترسون :

توضیح: این الگوریتم از دو تابع Enter(); و Exit(); تشکیل یافته است. قبل از وارد شدن به ناحیه بحرانی هر پردازه باید تابع Enter(); را صدا زده و شماره پردازه خود را به عنوان آرگومان به آن بفرستد.این تابع باعث می شود که پردازه تا زمانی که بتواند بدون خطر وارد ناحیه بحرانی شود، منتظر باقی بماند. پس از انجام کارهای ناحیه بحرانی(دستکاری متغیر های مشترک)، پردازه تابع Exit(); را صدا می زند که نشان دهد کارش در ناحیه بحرانی تمام شده و سایر پردازش ها در صورت نیاز می توانند وارد ناحیه بحرانی بشوند. پردازش ها از دو متغیر مشترک turn و flag استفاده می کنند .

```
int  turn;
int  flag[2]={0};
Enter(int pid)
{
int other;
turn=pid;
flag[pid]=1;
other=1-pid;
while((turn=pid)&&(flag[other]= = 1));
}
-.-.-.
```

6- سمافورها(Semaphores-راهنما ها)

سمافور x یک متغییر صحیح می باشد که جدای از مقدار دهی اولیه،فقط از طریق دو عمل اتمیک استاندارد زیر قابل دسترسی است

$$wait() \rightarrow p() - 1$$

$$signal() \rightarrow v() - 2$$

تعریف ساده p و v به صورت زیر است.

Semaphore(x)	
P(x)	v(x)
While(x<=0); x--	x++

☐ سمافور ها را می توان برای مساله بخش بحرانی n پردازش استفاده کرد.

فرض کنید دو پردازش p_1 و p_2 به شکل زیر اجرا شوند(مقدار اولیه سمافور 1 می باشد)

p_1	p_2
$p(x);$	$p(x);$
$c++;$	$c--;$
$v(x);$	$v(x);$

فرضا اگر ابتدا پردازه p_1 بخواهد اجرا شود با تابع p مقدار سمافور را صفر می کند و بعد وارد ناحیه بحرانی می شود(جهت دستکاری متغییر مشترک) و در انتها با دستور v مقدار سمافور را یک می کند، که قبل از یک شدن مقدار سمافور، اگر پردازه p_2 می خواست وارد ناحیه بحرانی شود در حلقه $while(x <= 0)$ گیر می افتاد.(تا لحظه ای که پردازه p_1 دستور v را اجرا کند و دوباره مقدار سمافور یک شود)

☐ اگر تعریف p و v به صورت داده شده باشد و سمافور x فقط در بردارنده یک مقدار باشد مشکل **Busy waiting** این جا نیز بر قرار است. برای رفع این مشکل تعریف سمافور ، p و v را کاملتر می کنیم. در این حالت زمانی که پروسس، اجازه ورود به ناحیه بحرانی اش را ندارد بلوکه یا مسدود می شود(به جای آنکه در یک حلقه **while** چرخ بزند)، بدین ترتیب آن پروسس به حالت تعلیق می رودتا هنگامی که پروسس دیگری آن را بیدار (**wakeup**) کند

☐ساختار واقعی سمافور، p و v به صورت زیر است.

```
Struct semaphore
{
List of process l;
Int value:
}
```

Semaphore x;	
P(x)	v(x)
x.value - -;	x.value++;
if(x.value<0) begin	if(x.value<=0)begin
add process to x.l;	remove a process p from x.l;
block the process;	wakeup(p);
end;	end;

توضیح: وقتی پردازشی عمل p را اجرا کرده و سمافور را غیر مثبت می یابد، باید صبر کند. اما به جای حلقه **while**، پردازش در صف انتظار مربوط به آن سمافور قرار داده می شود و حالت پردازش به **block** تغییر می یابد. سپس کنترل به زمانبند **CPU** منتقل می گردد تا پردازش دیگری را برای اجرا، انتخاب کند. فرایندی که **block** شده بر اثر اجرای عمل p توسط فرایند دیگر از سر گرفته می شود. این فرایند توسط یک عمل **wakeup** بیدار شده و از حالت انتظار به حالت آماده می رود و در صف **Ready** قرار داده می شود.

- در ساختار جدید مقدار سمافور می تواند منفی باشد، که قدر مطلق این مقدار برابر تعداد پردازه هائی است که منتظر این سمافور هستند
- اگر در هنگام اجرای دستور $wakeup(x)$ پردازش x در حالت بلوکه نباشد، این دستور هیچ عملی خاصی را انجام نمی دهد و اثری ندارد.

مسئله تولید کننده-مصرف کننده: دو پردازه تولید کننده و مصرف کننده وجود دارد که به طور همروند اجرا می شوند، تولید کننده یک سری اقلام اطلاعاتی را تولید می کند و در یک بافر که اندازه آن محدود است قرار می دهد. پردازه مصرف کننده این اقلام اطلاعاتی را از بافر برداشته و اجرا می کند.

```
Int  Buffer[max];
Int count=0;
Producer(){
While(1){
Produce an item;
Buffer[count]=item;
Count ++;
}
Concumer(){
Count--;
Item=Buffer[count];
Consum an item;
}
```

نکته: دستورات $count++$ و $count--$ به صورت زیر عمل می کنند.

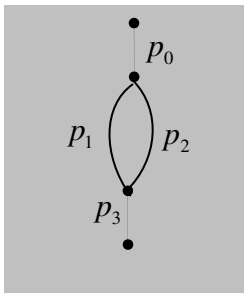
Count--	Count++
Mov Ax,Count;	Mov Ax,Count;
Sub Ax,1;	Add Ax,1;
Mov Count,Ax;	Mov Count,Ax;

- محدودیت هائی که در این مسئله وجود دارند عبارتند از:
- اگر بافر پر باشد و تولید کننده بخواهد یک قلم اطلا عاتی را در بافر قرار دهد، می بایست صبر کند تا بافر خالی شود.
- اگر بافر خالی باشد ومصرف کننده بخواهد یک قلم اطلا عاتی را از بافر بردارد، می بایست صبر کند تا اطلاعاتی در بافر قرار داده شود.
- مشکل دیگر مربوط به متغییر مشترک **count** می باشد، به طور مثال فرض کنید $count = 3$ باشد، و پردازه مصرف کننده بخواهد یک قلم اطلاعاتی رااز بافر بردارد، در این صورت اگر این پردازه به دستور Mov Ax,Count; برسد سه را در **Ax** قرار می دهد، حال قبل از اینکه مقدار تغییر یافته **count** یعنی دو در **Ax** نوشته شود، **Context switching** رخ می دهد و پردازه تولید کننده اجرا می شود.این پردازه مقدار **count** را سه می بیند، و بنا براین سه را در **Ax** قرار می دهد. اگر پردازه تولید کننده ابتدا، مقدار تغییر یافته **count** را بنویسد و بعد از آن پردازه مصرف کننده مقدار تغییر یافته **count** را بنویسد، مقدار **count**، دو خواهد شد و در حالت بر عکس مقدار نهائی چهار خواهد شد. که هر دو حالت نادرست هستند، زیرا مقدار نهائی باید سه باشد، پس این مشکلات می بایست حل شوند. با استفاده از سمافور ها می توانیم برنامه تولید کننده و مصرف کننده را به شکل زیر بنویسیم.

```
Semaphor  mutex=1, Count=0;
Semaphor  empty=max;
Semaphor  full=0;
Producer(){
While(1){
Produce an item;
P(empty);
P(mutex);
Buffer[count]= item;
Count ++;
V(mutex);
```

```
Consumer(){
 while(1){
P(full);
 P(mutex);
Count - -;
 item=Buffer[count];
 V(mutex);
 V(empty);
  consum the item;
  }}
```

مثال: فرض کنید پردازه های p_3, p_2, p_1, p_0 می خواهند به صورت هم روند اجرا شوند. با استفاده از سمافور ها دستوراتی بنویسید، که هماهنگی بین پردازه ها را به صورت شکل فوق فراهم کنند



حل:

Semaphor $S_{12} = 0$ Semaphore $S_3 = 0$

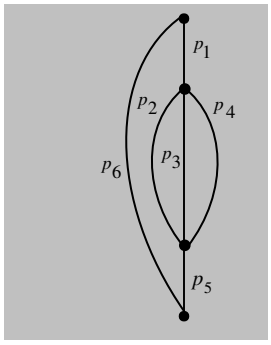
Cobegin( ){

$p_0(), p_1(), p_2(), p_3()$

}

$P_0()\{$	$P_1()\{$	$P_2()\{$	$P_3()\{$
$\vdots$	$P(S_{12})$	$P(S_{12})$	$P(S_3)$
$V(S_{12})$	$\vdots$	$\vdots$	$P(S_3)$
$V(S_{12})$	$V(S_3)$	$V(S_3)$	$\vdots$
$\}$	$\}$	$\}$	$\}$

مثال: فرض کنید پردازه های $p_6, p_5, p_3, p_2, p_1, p_1$ می خواهند به صورت هم روند اجرا شوند. با استفاده از سمافور ها دستوراتی بنویسید، که هماهنگی بین پردازه ها را به صورت شکل فوق فراهم کنند



حل:

$semaphor\ s_1 = 0\ , s_{234} = 0;$

Cobegin( ){

$p_1(), p_2(), p_3(), p_4(), p_5(), p_6()$

}

$p_1()\{$	$p_2()\{$	$p_3()\{$	$p_4()\{$	$p_5()\{$	$p_6()\{$
$\vdots$	$P(s_1)$	$P(s_1)$	$P(s_1)$	$P(s_{234});$	$\vdots$
$V(s_1);$	$\vdots$	$\vdots$	$\vdots$	$P(s_{234});$	$comands$
$V(s_1);$	$V(s_{234});$	$V(s_{234});$	$V(s_{234});$	$P(s_{234});$	$\vdots$
$V(s_1);$	$\}$	$\}$	$\}$	$\vdots$	$\}$
$\}$				$\}$	

پایان جلسه پنجم

جلسه ششم

سمافورها می توانند سبب بن بست شوند دو پردازه p_0 , p_1 را در نظر بگرید.

فرض کنید p_0 دستور $p(a);$ را اجرا کرده باشد، در همین لحظه عمل تعویض متن رخ دهد و پردازه p_1 دستور $p(b);$ را اجرا کند، حال اگر پرداز p_1 دستور $p(a);$ را اجرا کند، یا پردازه p_0 دستور $p(b);$ را اجرا کند، هر دو بلاک می شوند(هر دو منتظر هم باقی می مانند)، در این حالت می گوئیم بین پردازه ها بن بست رخ داده است، پس سمافور ها می توانند سبب بن بست شوند.

$Semaphor\ a=1, b=1;$

$P_0:$	$p_1:$
$p(a);$	$p(b);$
$p(b);$	$p(a);$
$\vdots$	$\vdots$
$v(a);$	$v(b);$
$v(b);$	$v(a);$

7- مانیتور(monitor):

سمافور ها با همه مزایائی که دارند،پیچیده می باشند به عبارتی، در استفاده از سمافورها می بایست خیلی دقت کرد، زیرا یک دستور v اضافی سبب مشکل می شود. مانیتور ها ابزارهای سطح بالائی هستند که این مشکل را ندارند، مانیتور یک ساختمان داده ای شامل توابع،متغیر های معمولی، متغیر های شرطی می باشند، که این متغیر های شرطی فقط از طریق دستورات **wait** و **signal** قابل دستیابی هستند، اگر پردازه ای در داخل مانیتور دستور **wait** را صادر کند، آن پردازه در لیست پردازه های آن متغیر شرطی قرار گرفته، و بلاک می شود و پردازه های دیگر می توانند وارد مانیتور شوند، یکی از توابع داخل مانیتور را اجرا کنند. در هر لحظه فقط یک پردازه می تواند در داخل مانیتور باشد. یعنی فقط یک پردازه می تواند در هر آن در حال اجرای توابع داخل مانیتور باشد.

- متغیر های شرطی، مقدار دهی نمی شوند و مقدار نمی گیرند.
- دستور **signal** سبب می شود اگر پردازه ای در صف بلاک آن متغیر شرطی باشد، بیدار شود و وارد مانیتور شده و اجرایش را از محلی که قبلا بلاک شده بود، ادامه دهد.
- کنترل ورود به ناحیه بحرانی را به هنگام استفاده از مانیتور، کامپایلر انجام می دهد، به طوری که کامپایلر اجازه نمی دهد بیش از یک پردازه در مانیتور اجرا شود، پس بهتر است نواحی بحرانی در مانیتور نوشته شود.

مسئله تولید کننده-مصرف کننده با استفاده از مانیتور

```
Monitor  Tolid - Masraf
Condition  full , empty

garardadan ( ){                Bardash tan( ){                producer ( ){                consumer ( ){
if ( count == max)             if ( count == 0)               while (1){                   while (1){
wait ( full );                   wait ( empty );              produce an item ;            Tolid - masraf .bardash tan();
Buffer [ count ] = item ;      count --;                      tolid - masraf .garardadan ()   consum item ;
count ++;                        item = Buffer [ count ];       ⋮                            ⋮
if ( count == 1)                 if ( count == max - 1)       }}                           }}
signal ( empty )               signal ( full );
}                            }
```

☐ مانیتور را می بایست حتما کامپایلر پشتیبانی کند، ولی برای استفاده سمافور لزومی به پشتیبانی از طرف کامپایلر نیست، فقط کافی است سیستم عامل سمافور را پشتیبانی کند، که در این صورت می توان با استفاده از دستورات p,v پیاده سازی کرد.

بن بست پردازه ها(Process Deadlocks)

اگر مجموعه ای از پردازه ها در سیستم منتظر وقوع حادثه ای باشند که توسط دیگری انجام شود و هیچ کاری در سیستم پیش نرود، گوئیم سیستم دچار بن بست شده است.

شرایط وقوع بن بست: برای رخ دادن یک بن بست هر چهار شرط زیر باید برقرار باشند.

1- انحصار متقابل(Mutual Exclusion)

2- گرفتن و منتظر ماندن(Hold and wait)

3-عدم پس گرفتن(انحصاری بودن)(No preemption)

4- انتظار چرخشی(Circular Wait)

انحصار متقابل بدین معناست که منبع یا منابع درهر آن، فقط توسط یک پردازه قابل استفاده باشند.

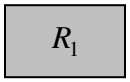
گرفتن و منتظر ماندن یعنی پردازه یک سری منابع مورد نیازش را در اختیار گرفته، و منتظر منابعی است که در اختیار دیگر پردازه ها است

عدم پس گرفتن بدین معناست که به اجبار نمی توان منبع یا منابعی را از پردازه پس گرفت.

انتظار چرخشی یعنی بایستی مجموعه ای از پردازش ها $\{p_0, p_1, \cdots p_n\}$وجود داشته باشد به طوری که p_0 منتظر منبعی از p_1, p_1 منتظر منبع از p_2 و ... p_n منتظر منبعی از p_0 باشد.

روش توصیف بن بست: استفاده از گراف تخصیص منابع(Resource Graph) $\leftarrow G(V,E)$

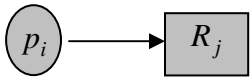
منابع = R= مستطیل



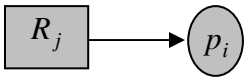
پردازه =P= دایره

در گراف تخصیص منابع، گره ها از نوع منابع یا پردازه اند، که منابع با مستطیل نمایش داده می شوند و تعداد نمونه های آن با نقطه داخل آن مشخص می شوند و پردازه ها با دایره مشخص می شوند.

یال های گراف تخصیص منابع جهت دار می باشند که یا از پردازه به منابع می باشد(بدین معناست که پردازه p_i منتظر منبع R_j است) به این شکل.



یا از منبع به پردازه می باشد(بدین معنا است که یک نمونه از منبع R_j در اختیار پردازه p_i می باشد) به این شکل.



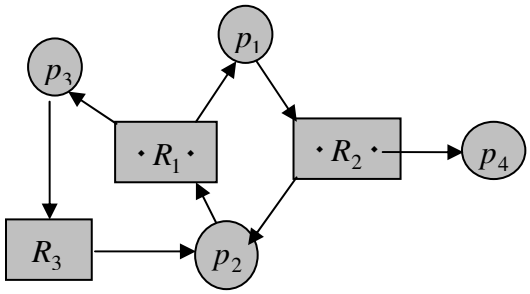
☐می توا اثبات کرد که اگر گراف دارای هیچ سیکلی(حلقه یا $loop$) نباشد، هیچ پردازشی در سیستم در بنبست نخواهد بود.

☐ اگر گراف دارای سیکلی باشد، احتمال دارد بن بست وجود داشته باشد. پس وجود حلقه در گراف شرط لازم برای بن بست است و نه شرط کافی

☐ اگر در گراف هر منبع دقیقا یک نمونه داشته باشد،آنگاه اگر گراف حلقه داشته باشد، بدین معناست که حتما بن بست رخ داده است، ولی اگر هر نوع منبع نمونه های متعددی داشته باشد، انگاه حلقه الزاما به معنای وقوع بن بست نیست

☐ اگر در گراف حلقه ای وجود نداشته باشد، آنگاه سیستم در حالت بن بست نیست

مثال: آیا گراف زیر در بن بست قرار دارد.

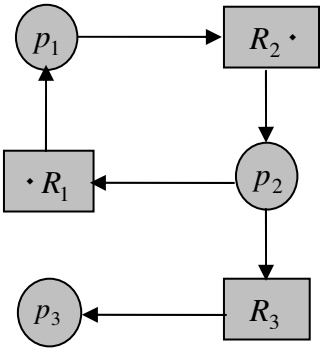


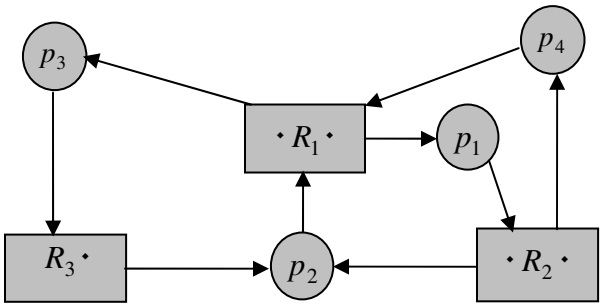
حل: خیر

توضیح:در شکل گفته شده در گراف حلقه وجود دارد ولی سیستم(پردازه ها) در بن بست نیستند، زیرا پردازه p_4 به منابع دیگری نیاز ندارد و بنابراین اجرایش را به اتمام رسانده و منبع R_2 را رها می سازد با رها ساختن منبع R_2 بن بست شکسته میشود. منبع R_2 به پردازه ی p_1 تخصیص داده میشود پردازه P_1 تمام منابع موردنیازش را در اختیار دارد، اجرایش را به اتمام رسانده و منابع را آزاد می سازد. در این حال منبع R_1 را به پردازه p_2 داده، پردازه p_2 نیز با در اختیار داشتن تمام منابع مورد نیازش، اجرایش را به پایان رسانده و تمام منابع را آزاد می سازد. که یکی از منابع R_3 است. منبع R_3 به پردازه p_3 تخصیص داده شده و این پردازه با در اختیار داشتن منابع اجرایش را به پایان می رساند.

مثال: آیا گراف های زیر در بن بست قرار دارند

حل:بله در هر دو بن بست وجود دارد.





فرایند استفاده از منبع:

1- در خواست منبع(Request): اگر پردازه ای به منبعی نیاز داشته باشد، دستور سیستمی درخواست منبع را می دهد، که اگر آن منبع آزاد باشد، به پردازه درخواست کننده تخصیص داده می شود، و گرنه پردازه می بایست منتظر منبع باشد

2- بکارگیری منبع(allocation): در صورت وجود منبع درخواست شده، منبع به پردازه درخواست کننده توسط سیستم تخصیص داده می شود.

3- آزاد کردن منبع(Release): پردازه پس از استفاده از منبع تخصیص یافته، منبع را به سیستم برمی رداند.

☐ در خواست منبع و آزاد کردن آن، توسط دستورات سیستم عامل(System Call) انجام می شود. مثلا برای فایل دستورات *open* و *close* و برای سمافور *wait* و *signal* انجام می شود. و همواره سیستم کلیه درخواست ها، تخصیص ها و آزاد شدن منابع را تحت نظر دارد

پایان جلسه ششم.

جلسه هفتم

روش های اداره بن بست(Dead lock handling)

4 روش برخورد با بن بست وجود دارد.

1- **پیشگیری از بن بست:** برای پیشگیری از بن بست می بایست کاری کرد که یکی از شرایط چهارگانه وقوع بن بست نقض شود. بدین ترتیب هرگز بن بست رخ نخواهد داد.

- **نقض انحصار متقابل :** اگر از منابعی به صورت اشتراکی استفاده کنیم بن بست هیچگاه رخ نمی دهد، نمونه آن فایل های فقط خواندنی هستند که می توانند همزمان توسط چندین پردازش استفاده شوند. البته این روش برای یک سری از منابع که ذاتا ماهیت غیر اشتراکی دارند قابل استفاده نیست مثل چاپگر. البته با ***spool*** کردن خروجی چاپگر چندین پروسس می توانند در یک زمان خروجی های خود را تولید کنند ولی تمام دستگاه ها را نمی توان ***spool*** کرد مثل جدول پروسس.
- **نقضِ گرفتن و منتظر ماندن :** برای نقض این شرط، باید مانع از ایجاد موقعیتی شد که در آن یک پردازه، منبعی را در اختیار گیرد و تقاضای منبع دیگری را نماید. رسیدن به این هدف با دو روش امکان پذیر است.

A. همه منابع مورد نیاز پردازه ها در ابتدای شروع اجرای پردازه اگر در دسترس باشند به آن اختصاص داده شوند وگرنه اختصاص داده نشوند به عبارتی یا همه منابع اختصاص داده شوند یا هیچکدام اختصاص داده نشوند. به عنوان مثال پردازه ای که می بایست داده ها را از نوار مغناطیس خوانده سپس آنها را بر روی دیسکی کپی کرده و آنها را مرتب کرده و چاپ کند، می بایست در ابتدا چاپگر را که در انتهای کارش نیاز دارد به دست آورد ، پس از چاپگر به درستی استفاده نشده است ، زیرا در همین حین که که چاپگر بدون کار در اختیار این پردازه بوده ، پردازه دیگری می توانست از این استفاده کند. پس عیب این پروتکل کاهش بهره وری سیستم است.

B. هر پردازه ای که یک سری منابع را در اختیار دارد، و طلب منابع دیگری می کند می بایست منابع در اختیارش را آزاد سازد، سپس تمام منابع را با هم تحویل بگیرد، مشکل این روش قحط زدگی (گرسنگی) است، یعنی پردازه ای که منابع متعددی نیاز دارد بایستی به طور نامعین در انتظار باشد چرا که به احتمال زیاد یکی از منابع مورد نیازش همواره توسط پردزاش دیگری استفاده شده است.

- **نقض عدم پس گرفتن:** جهت تحقق این شرط دو راه حل وجود دارد.

A. یکی این است که اگر پردازه ای درخواست منابعی را داد که آن منابع آزاد نباشند،تمام منابع در اختیار پردازه ی درخواست کننده پس گرفته شوند.

B. راه حل دیگر آن است که بررسی شود که آیا پردازه های منتظر تمام منابع مورد نیاز پردازه درخواست کننده را دارند یا نه، که اگر داشته باشند منابع از پردازه های منتظر گرفته شده و به پردازه درخواست کننده انتساب یابد و اگرپردازه یا پردازه های منتظر، منابع موردنیاز پردازه ی درخواست کننده را نداشته باشند، خود پردازه ی درخواست کننده باید منتظر بماند وممکن است در حین انتظار منابع در اختیارش نیز از وی گرفته شوند وبه پردازه های دیگری انتساب یابند.

- **نقض انتظار چرخشی:**

جهت نقض انتظار چرخشی می بایست منابع را شماره گذاری کرد به طوری که هر پردازه بتواند منابع را در جهت صعودی شماره هایشان درخواست کند. به عنوان مثال اگر پردازه ای منبع شماره 3 را در اختیار داشته باشد، منبع شماره ی 1 را نمی تواند درخواست بکند، ولی می تواند منبع شماره ی 5 را درخواست کند. در این درخواست برای یک منبع می توان چندین نمونه از آن منبع را طلب کرد. شماره گذاری منابع می بایست بر اساس ترتیب نیاز به منابع صورت گیرد.به عنوان مثال، در مثال قبل می بایست شماره ی نوار مغناطیسی کمتر از شماره ی دیسک و شماره ی دیسک کمتر ازشماره ی چاپگر باشد. جهت اثبات اینکه روش گفته شده انتظار چرخشی را نقض می کند از برهان خلف استفاده می کنیم. فرض کنیم با اعمال این

شرط باز هم انتظار چرخشی داشته باشیم. در این صورت اگر شماره منبع در اختیار پردازه ی p_{i-1} ، $f(R_{i-1})$ باشد،این پردازه در صورتی منتظر P_i است که $f(R_{i-1}) < f(R_i)$ با شد و با تعمیم این رابطه به رابطه ی می رسیم. که می دانیم

$$f(R_0) < f(R_1) < ... < f(R_n) < f(R_0) \qquad f(R_n) < f(R_0)$$

نیست و برهان خلف اثبات می شود.

عیب روش پیشگیری از بن بست، بهره وری پایین منابع و کاهش توان عملیاتی سیستم می باشد .

2- اجتناب از بن بست:

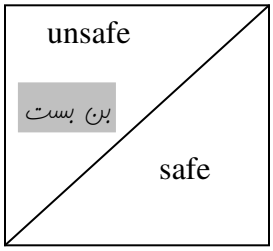
در این روش با توجه به اطلاعاتی نظیر حداکثر نیاز پردازه ها به منابع و منابع تخصیص یافته به پردازه ها و موجودی، وقتی پردازه ای درخواست یک سری منابع را داشته باشد، اگر منابع موجود باشند، بررسی می کنیم که آیا با اجابت این درخواست سیستم به حالت امن میرود یا نه. اگر سیستم به حالت امن برود درخواست تخصیص داده می شود و گرنه از درخواست اجتناب می شود .

حالت امن حالتی است که پردازه ها میتوانند به ترتیب خاص منابع مورد نیازشان را گرفته و اجرایشان را با موفقیت پشت سر بگذرانند.

تعریف دیگر: حالتی است که در آن یک ترتیب امن(Safe sequence) از پردازه ها وجود داشته باشد .

تعریف دیگر: اگر سیستم بتواند منابع مورد درخواست را به ترتیبی تخصیص دهد که از بروز بن بست اجتناب شود گوئیم آن سیستم د رحالت امن است

Safe sequence.(ترتیب امن) ترتیبی است از پردازه ها که می توانند با ترتیب معینی از تحویل گرفتن منابع،اجرایشان را خاتمه دهند.



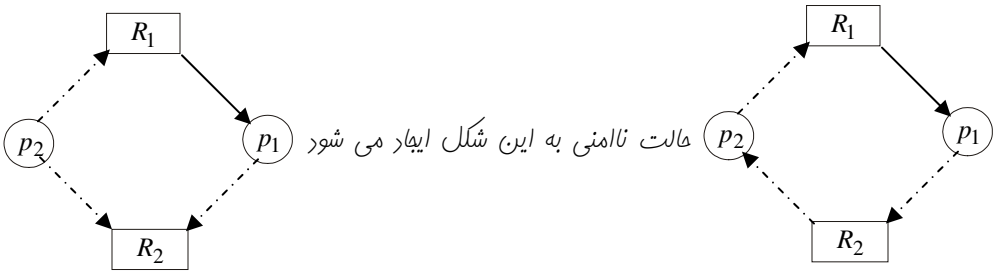
برای اجتناب از بن بست دو راه وجود دارد:

1- از هر منبع فقط یک نمونه وجود داشته باشد: از گرافی شبیه گراف تخصیص منابع استفاده می شود.

2- از هر منبع بیش از یک نمونه وجود داشته باشد. از الگوریتم بانکداران استفاده می شود.

گراف تخصیص منبع :این گراف شبیه گراف تخصیص منابع است که علاوه بر یال های انتظار وتخصیص شامل یک یال ادعا (claim eddge) می باشد که با نقطه چین نمایش داده می شود.به طوریکه p_i - - - ▶ R_i بدین معناست که ممکن است در آینده پردازه ی p_i درخواست منبع R_i را داشته باشد.هریال تخصیص پس از پایان تخصیص تبدیل به یال ادعا می شود.

با توجه به شکل اگر p_2 در لحظه ی فعلی درخواست R_2 را داشته باشد، این درخواست به وی اعطا نمی شود، زیرا سیستم به حالت نا امن می رود به طوری‌که ممکن است در آینده P_2 طلب R_1 و P_1 طلب R_2 را داشته باشد، که بن بست رخ می دهد



اگردر گراف بالایی P_1 درخواست R_2 را داشته باشد،آیا این درخواست اجابت شود یا خیر؟

بله،اجابت می شود، زیرا در آینده ایجاد حلقه غیر ممکن است.

الگوریتم بانکداران: این الگوریتم اولین بار توسط دیجسترا ارائه شد ، و به نام الگوریتم بانکدار معروف گردید، چرا که این الگوریتم شبیه، رفتار یک بانکدار شهر کوچک با مشتریانش طراحی شده است، یک بانکدار هرگز تمام سرمایه خودش را به مشتریان تخصیص نمی دهد و طوری عمل می کنند که بتواند کلیه نیاز های مشتریانش را بر آورده کند.

ساختمان داده های مورد نیاز(n تعداد پردازه ها وm تعداد منابع)

1. $Max_{n\times m}$

- $Max[i][j]=k$ بدین معناست که پردازه p_i ، جهت اجرایش به k نمونه از منبع R_j نیاز دارد

2. $Alocation_{n\times m}$

- $Alocation[i][j]=k$ به این معناست که در حال حاضر k نمونه از منبع R_j در اختیار پردازه p_i ، می باشد.

3. $Need_{n\times m}$

- $Need[i][j]=k$ به این معناست که پردازه p_i جهت ادامه کارش به k نمونه از منبع R_j نیاز دارد.

- بدیهی است که $Need[i][j]=Max[i][j]-Alocation[i][j]$ می باشد
- بردار $Available$ (در دسترس) به طول m تعداد منابع آزاد از هر نوع را نشان می دهد. مثلا $Available[j]=k$ باشد یعنی از منبع نوع R_j به تعداد k نمونه در دسترس وجود دارد.

اگر $Request_i$ بردار درخواست منابع ، پردازه p_i باشد الگوریتم به صورت زیر نوشته می شود.

1- اگر $Need_i < Request_i$ (یعنی سطر مربوط به پردازه p_i در ماتریس) باشد،این درخواست غیر قانونی است ، و بررسی نمی شود(یعنی پایان الگوریتم)

- دو بردار x و y را به طول n در نظر بگیرید. می گوئیم $x\le y$ اگر و فقط اگر $x[i]\le y[i]$ باشد به ازاء $i=1,2,\cdots,n$. مثلا $(0,4,2,3)\le(2,8,3,5)$

2- اگر $Request_i > Available$ باشدبدین معناست که منابع کافی جهت تخصیص به پردازه p_i ، وجود ندارد، و p_i می بایست منتظر بماند.

3- وانمود می کنیم که منابع مورد درخواست پردازه p_i ، به وی اعطاء می شود، بنابراین ساختمان داده ها به صورت زیر تغییر میی یابند.

$Alocation_i += Request_i$
$Need -= Request_i$
$Available -= Request_i$

4- فراخوانی الگوریتم امنیت: اگر حالت سیستم ناامن باشد ساختمان داده های تغییر یافته در قسمت 3 به حالت اول بر گردانده می شود(اجتناب از بن بست)

5- پایان

☹ مرتبه اجرای الگوریتم بانکدار $m\times n^2$ می باشد(n تعداد پردازش ها و m تعداد انواع منابع است)

الگوریتم امنیت

استفاده از بردار های work به طول m (متناظر با منابع) و Finish به طول n (متناظر با پردازه ها)

1- work=Available و برای Finish[i]=False : i=1…n

2- به ازای i=1,…,n (به ازای تمام مقایر پردازه ها) مقدار i را چنان بیابید که، Finish[i]=Fals و $Need_i \le work$ اگر چنین i ای پیدا نشد برو به مرحله ی 4

(در این مرحله به دنبال پردازه ای می گردیم که به پایان نرسیده باشد و با منابع موجود بتوان اجرایش را به پایان رساند)

3- زمانی به مرحله سه می آییم که پردازه ای پیدا شود که اجرایش تمام نشده و با استفاده از منابع موجود می تواند نیاز هایش را بر آورده کرده و اجرایش را به اتمام برساند بنابراین پس از اتمام اجرایش منابع در اختیارش را آزاد می سازد. $work += Alocation$, $finish[i] = true$

4- اگر برای تمام i ها (i=1…n): Finish[i]=true باشد ، بدین معناست که تمام پردازه ها می توانند با منابع موجود اجرایشان را با موفقیت پشت سر بگذرانند و سیستم در حالت امن قرار دارد ولی اگر حداقل یک i پیدا شود که Finish[i]=false باشد بدین معناست که با منابع موجود این پر دازه نمی تواند اجرایش را به اتمام برساند و سیستم در حالت ناامن قرار دارد.

مثال. فرض کنید سیستمی با چهار نوع منبع R_1 , R_2 , R_3 , R_4 که هر کدام به ترتیب دارای 8 , 5 , 9 , 7 نمونه می باشند، موجود باشد اگر در این سیستم 5 پردازه ی p_1 الی p_5 را داشته باشیم، با توجه به ماتریس های max و *Alocation* تشخیص دهید سیستم در حالت امن است یا نه؟

process	R_1	R_2	R_3	R_4
p_1	3	2	1	4
p_2	0	2	5	2
p_3	5	1	0	5
p_4	1	5	3	0
p_5	3	0	3	3

max

process	R_1	R_2	R_3	R_4
p_1	2	0	1	1
p_2	0	1	2	1
p_3	4	0	0	3
p_4	0	2	1	0
p_5	1	0	3	3
sum	7	3	7	5

Alocation

حل. ابتدا ماتریس *Need* را می سازیم با توجه به $Need = Max - Alocation$ خواهیم داشت .

process	R_1	R_2	R_3	R_4
p_1	1	2	0	3
p_2	0	1	3	1
p_3	1	1	0	2
p_4	1	3	2	0
p_5	2	0	0	3

Need

حال بردار *Avalable* را با توجه به فرمول $Avalable =$ موجودی اولیه $- sum$ ایجاد می کنیم و بر دار *finish* و *work* نیز در ابتدا به شکل زیر می باشند.

	R_1	R_2	R_3	R_4
Available	1	2	2	2

	p_1	p_2	p_3	p_4	p_5
finish	*f*	*f*	*f*	*f*	*f*

	R_1	R_2	R_3	R_4
work	1	2	2	2

پردازه هائی را که *false* هستند با *work* مقایسه می کنیم.(مقایسه سطر های ماتریس *Need* با *work*)، که در اولین مقایسه ملاحظه می شود که فقط پردازه p_3 کوچکتر است و می تواند اجرایش را با موفقیت انجام دهد و پس از اجرا منابع را بر گرداند، کار را به همین ترتیب ادامه می دهیم.

	R_1	R_2	R_3	R_4
work	5	2	2	5

	p_1	p_2	p_3	p_4	p_5
finish	*f*	*f*	*T*	*f*	*f*

	R_1	R_2	R_3	R_4
work	7	2	3	6

	p_1	p_2	p_3	p_4	p_5
finish	*T*	*f*	*T*	*f*	*f*

	R_1	R_2	R_3	R_4
work	7	3	5	7

	p_1	p_2	p_3	p_4	p_5
finish	*T*	*T*	*T*	*f*	*f*

	R_1	R_2	R_3	R_4
work	7	5	6	7

	p_1	p_2	p_3	p_4	p_5
finish	*T*	*T*	*T*	*T*	*f*

	R_1	R_2	R_3	R_4
work	8	5	9	7

	p_1	p_2	p_3	p_4	p_5
finish	*T*	*T*	*T*	*T*	*T*

ملاحظه می شود که سیستم در حالت امن است وترتیب اجرا به صورت p_3, p_1, p_2, p_4, p_5 می باشد

در همین سئوال پردازه p_3 درخواست یک نمونه از منبع R_3 را داشته باشد آیا این درخواست پاسخ داده شود یا خیر؟

مثال.سیستمی با پنج فرایند p_0 الی p_4 و سه نوع منبع C, B, A را در نظر بگیرید منبع نوع A دارای 10 نمونه، منبع نوع B دارای 5 نمونه و منبع نوع C دارای 8 نمونه می باشد، فرض کنید در زمان t_0 وضعیت سیستم به صورت زیر باشد

process	*A*	*B*	*C*
p_0	7	5	3
p_1	3	2	2
p_2	9	0	2
p_3	2	2	2
p_4	4	3	3

max

process	*A*	*B*	*C*
p_0	0	1	0
p_1	2	0	0
p_2	3	0	2
p_3	2	1	1
p_4	0	0	2

Allocation

الف. ماتریس *Need* را بیابید

ب. آیا در لحظه فعلی سیستم در حالت امن است یا نه؟ اگر در حالت امن است توالی امن را بیابید

ج. فرض کنید در لحظه t_0 فرایند p_1 یک نمونه از منبع A و دو نمونه دیگر از منبع C درخواست کند آیا این درخواست فورا پاسخ داده شود یا خیر.

د. اگر پردازه p_4 در خواست سه نمونه از منبع A و سه نمونه دیگر از منبع B را درخواست کند، آیا این درخواست می بایست بر آورده شود یا خیر.

process	A	B	C
p_0	7	4	3
p_1	1	2	2
p_2	6	0	0
p_3	0	1	1
p_4	4	3	1

Need

حل.

الف. با توجه به $Need = \max - Allocation$ داریم

ب. ابتدا بردار *Available* را می سازیم

A (آزاد) منابع موجود $= 10-(2+3+2) = 3$ $\quad$ B (آزاد) منابع موجود $= 5-(1+1) = 3$ $\quad$ C (آزاد) منابع موجود $= 7-(2+1+2) = 2$

پس بردار منابع در دسترس(*Available*) به صورت (3,3,2) می باشد که با توجه به این بردار مسئله را حل می کنیم.

منابع آزاد $(3,3,2) \xrightarrow{p_1} (5,3,2) \xrightarrow{p_3} (7,4,3) \xrightarrow{p_4} (7,4,5) \xrightarrow{p_2} (10,4,7) \xrightarrow{p_0} (10,5,7)$

همانطور که مشاهده می شود ترتیب اجراء $< p_1, p_3, p_4, p_2, p_0 >$ یک ترتیب امن است . لذا سیستم در حالت امن قرار دارد

نکته:اگر در یک لحظه چندین پروسس شرط لازم را برای اجرا شدن داشته باشند، اهمیتی ندارد که کدام یک جهت تخصیص منابع انتخاب شود چرا که در هر حال آن پروسس پس از تخصیص منابع مورد نیاز و تمام شدن،کل منابع خود را آزاد می کند.

ب. اگر درخواست فوق را برآورده سازیم ماتریس های *Allocation* و *Need* به صورت زیر در خواهند آمد

process	A	B	C
p_0	0	1	0
p_1	❸	0	❷
p_2	3	0	2
p_3	2	1	1
p_4	0	0	2

Allocation

process	A	B	C
p_0	7	4	3
p_1	⓪	2	⓪
p_2	6	0	0
p_3	0	1	1
p_4	4	3	1

Need

منابع آزاد $(2,3,0) \xrightarrow{p_1} (5,3,2) \xrightarrow{p_3} (7,4,3) \xrightarrow{p_4} (7,4,5) \xrightarrow{p_0} (7,5,5) \xrightarrow{p_2} (10,5,7)$

همانطور که مشاهده می شود ترتیب اجراء $< p_1, p_3, p_4, p_0, p_2 >$ یک ترتیب امن است . لذا اگر به درخواست های p_1 در لحظه t_0 پاسخ بگوئیم مطمئن هستیم که سیستم در حالت امن باقی خواهد ماند.

ب. اگر درخواست فوق را برآورده سازیم ماتریس های *Allocation* و *Need* به صورت زیر در خواهند آمد

process	*A*	*B*	*C*
p_0	7	4	3
p_1	1	2	2
p_2	6	0	0
p_3	0	1	1
p_4	❶	⓪	1

Need

process	*A*	*B*	*C*
p_0	0	1	0
p_1	2	0	0
p_2	3	0	2
p_3	2	1	1
p_4	❸	❸	2

Allocation

(0,0,2)منابع آزاد

حال اگر منابع آزاد(بردار *Available*) را با هر سطر ماتریس *Need* مقایسه کنیم، هیچ کدام از سطرهای ماتریس *Need* از بردار مذکور کمتر نیست لذا این وضیعت نا امن می باشد

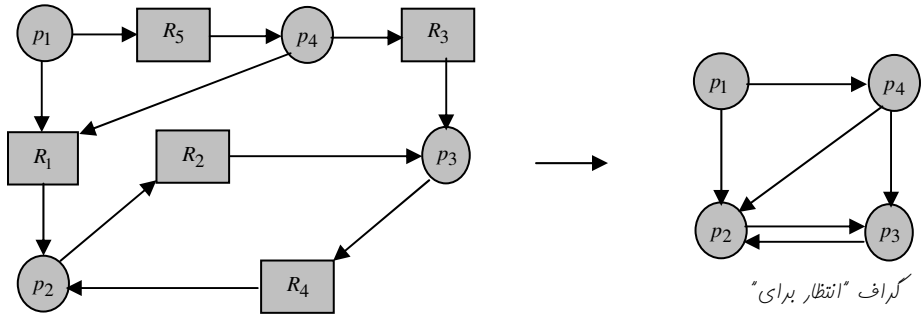
جلسه هشتم

3- روش تشخیص بن بست و بازیافت سیستم :

تشخیص بن بست برای حالت یک نمونه از هر منبع

در این روش از روی گراف تخصیص منبع، گراف انتظار($wait-for\ graph$) را بدست می آوریم. برای بدست آوردن گراف انتظار، گره های منبع را از گراف تخصیص حذف کرده و کمان های مناسبی را باهم ترکیب می کنیم.

مثال. فرض کنید گراف تخصیص منابعی به شکل زیر باشد، اولا گراف "انتظار برای" را رسم کنید ثانیا مشخص نمائید که سیستم دچار بن بست شده است یا نه.



گراف تخصیص منبع

گراف "انتظار برای"

چون حلقه وجود دارد سیستم دچار بن بست شده است

یک پیکان از پردازش p_i به p_j در گراف انتظار، نمایانگر این است که پردازش p_i درانتظار پردازش p_j است تا منبعی را که p_i به آن نیاز دارد، آزاد کند. سیستم عامل، گراف انتظار را نگهداری کرده و مرتبا آن را بررسی می کند. اگر در گراف انتظار حلقه وجود داشته باشد، آنگاه سیستم به بن بست رسیده است. الگوریتم تشخیص بن بست در گراف از مرتبه $O(n^2)$ میباشد که n تعداد رئوس گراف است.

تشخیص بن بست برای حالت چند نمونه از هر منبع

روش گراف انتظار برای این حالت قابل استفاده نیست. در حالتی که هر منبع چند نمونه دارد می بایست از روشی شبیه بانکداران استفاده کنیم. در این روش به جای ماتریس $Need$ از ماتریس $n\times m$ به نام $Request$ استفاده می شود که نیاز های فعلی هر پردازش را نشان می دهد. اگر $Request[i][j]=k$ باشد یعنی پردازش p_i ، k نمونه بیشتر از منبع R_j را درخواست کرده است. الگوریتم این روش به شکل زیر خواهد بود.

1- ابتدا بردار $work$ را مساوی $Available$ قرار میدهیم($work=Available$)اگر به ازای $i=0\cdots n$ ، $Allocation\ (i)\neq 0$ باشد پس $finish[i]=fals$ و گرنه $finish[i]=True$

2- به ازای $i=0\cdots n$ ، یک i چنان پیدا کنید که $finish\ [i]=fals$ باشد و $Request[i]>work$ (سطر مربوط به پردازه p_i در ماتریس $Request$) باشد. اگر چنین i پیدا نشد برو به مرحله 4 .

3- به ازای i پیدا شده در مرحله دو $finish[i]$ را برابر $True$ قرار می دهیم و $work$ را با $Allocation$ مربوطه جمع می کنیم ($work+=Alocation$)، زیرا پردازهای پیدا شده که با منابع موجود میتواند اجرایش را کامل کند و پس از خاتمه اجرا، منابع در اختیارش را آزاد سازد، دوباره برو به مرحله 2 .

4- اگر پردازه p_i یافت شود که $finish[i]=false$ باشد، بدین معناست که p_i در بن بست قرار دارد.(سیستم دچار بن بست شده است)

☐ پیچیدگی این الگوریتم $(m\times n)$ می باشد که m تعداد منابع است و n تعداد پردازه ها می باشد.

مثال. سیستمی با 5 پردازه $p_1\cdots p_5$ و سه نوع منبع A,B,C را در نظر می گیریم، A دارای 7 نمونه، منبع نوع B دارای 2 نمونه و c دارای 6 نمونه است فرض کنید در زمان t_0 وضعیت سیستم به صورت زیر باشد آیا سیستم در بن بست قرار دارد یا خیر؟

ب. فرض کنید p_3 نمونه دیگری از منبع C را درخواست کند آیا سیستم دچار بن بست می شودیا خیر؟

وضیعت اولیه سیستمَ

Process	A	B	C
p_1	0	1	0
p_2	2	0	0
p_3	3	0	3
p_4	2	1	1
p_5	0	0	2

Allocation َ

Process	A	B	C
p_1	0	0	0
p_2	2	0	2
p_3	0	0	0
p_4	1	0	0
p_5	0	0	2

Request َ

حل: بردار Available َ برابر خواهد بود با : (0,0,0) ، پس داریم

	A	B	C
Work	0	0	0

	p_1	p_2	p_3	p_4	p_5
finish	f	f	f	f	f

	A	B	C
Work	0	1	0

	p_1	p_2	p_3	p_4	p_5
finish	T	f	f	f	f

	A	B	C
Work	3	1	3

	p_1	p_2	p_3	p_4	p_5
finish	T	f	T	f	f

	A	B	C
Work	5	1	3

	p_1	p_2	p_3	p_4	p_5
finish	T	T	T	f	f

	A	B	C
Work	7	2	4

	p_1	p_2	p_3	p_4	p_5
finish	T	T	T	T	f

	A	B	C
Work	7	2	6

	p_1	p_2	p_3	p_4	p_5
finish	T	T	T	T	T

پس توالی $< p_1, p_3, p_2, p_4, p_5 >$ امکان پذیر است و سیستم در بن بست قرار ندارد.

ب. در این صورت ماتریس Request به شکل زیر می باشد. هر که می توان منابع p_1 را باز پس گرفت، ولی این تعداد منابع موجود برای انجام تقاضاهای هیچ پردازشی کافی نخواهد بود. لذا بن بستی شامل پردازش های p_2, p_3, p_4, p_5 پدید می آید.

Process	A	B	C
p_1	0	0	0
p_2	2	0	2
p_3	0	0	1
p_4	1	0	0
p_5	0	0	2

بن بست رخ می دهد

منابع $(0,0,0) \xrightarrow{p_1} (0,1,0)$

زمان فراخوانی الگوریتم تشخیص بن بست:

سئوال مهم این است که الگوریتم تشخیص بن بست را چه زمان هائی باید اجرا کنیم.

در یک حالت حد، میتوان در هر بار درخواستی که سریعا قابل اعطاء نمی باشد، این الگوریتم را اجرا کنیم. بدین ترتیب علاوه بر اینکه می توانیم مجموعه پردازش های موجود در بن بست را تشخیص دهیم، پردازش خاصی که باعث بن بست شده است نیز مشخص می شود. ولی از طرف دیگر با توجه به زمانگیر بودن الگوریتم های تشخیص بن بست، اجرای مکرر آنها باعث کاهش کارائی سیستم می گردد. یک روش کم هزینه تر آن است که الگوریتم مذکور را با پریود کمتری اجرا کنیم. مثلا در هر ساعت یکبار، یا هر بار که بهره وری *CPU* به زیر 40% برسد، یا هر بار که بار سیستم کم است. اگر این الگوریتم در زمانهای دلخواهی اجرا شود، ممکن است حلقه های بسیاری در گراف منبع وجود داشته باشند و بدین ترتیب در حالت کلی نمی توان گفت کدام پردازش باعث بن بست شده است.

ترمیم یا بازیافت سیستم:

یک روش برای ترمیم بن بست این است که بعد از اینکه بن بست تشخیص داده شد، به اپراتور خبر داده شود که سیستم دچار بن بست شده است که در این صورت اپراتور به صورت دستی سیستم را می بایست ترمیم کند. ولی اگر خود سیستم عامل بخواهد آن را باز یافت کند دو راه وجود دارد.

1- خاتمه دادن به پردازش ها:

- تمام پردازه های درگیر بن بست خاتمه داده شوند، که این راه حل هر چند سربار ناشی از اجرای متعدد الگوریتم های تشخیص بن بست را ندارد ولی ممکن است سیستم متحمل هزینه زیادی شود، چرا که ممکن است پردازه هائی دستورات زیادی را اجرا کرده باشند که بدین ترتیب می بایست دوباره از ابتدا اجرا شوند.
- روش دوم این است که هر بار یک پردازه انتخاب شده و خاتمه داده میشود، سپس الگوریتم کشف بن بست فراخوانی می شود. اگر سیستم دچار بن بست باشد، مجددا پردازه دیگری انتخاب شده و خاتمه داده می شود، تا جائی که سیستم از بن بست خارج شود. اشکال این روش سربار زمانی ناشی از الگوریتم کشف بن بست می باشد. انتخاب یک پردازه جهت خاتمه دادن می بایست بر اساس هزینه کمینه باشد، که هزینه کمینه می تواند به فاکتور های زیر بستگی داشته باشد.

1- نوع پردازه(دسته ای یا محاوره ای)

2- مدت زمانی که از اجرای پردازه سپری شده است

3- مدت زمانی که اجرای پردازه باقی مانده است

4- تعداد و نوع منابعی که در اختیار پردازه است

5- درصد استفاده پردازه از منابع

6- تعداد منابعی که برای کامل شدن نیاز دارد

2- پس گرفتن منابع:

در این روش منابعی از یک پردازش گرفته شده و در اختیار پردازش دیگری قرار داده می شود. برای این کار باید سه موضوع مشخص گردد.

الف. انتخاب منبع و پردازش های مورد نظر

ب. باز گرداندن به عقب(*Rollback*) یعنی پردازشی که منبع او پس گرفته شده، باید به حالت امنی به عقب برگردانده شود، تا بعدا از آن حالت مجددا اجرایش را از سر بگیرد.

ج.قحطی زدگی یعنی باید تضمین کرد که منابع همواره از یک پردزاش خاص بازپس گرفته نشود،چرا که در آن صورت اجرای آن پردازش مرتبا به تعویق می افتد

4- روش صرف نظر کردن از بن بست(الگوریتم استریخ *ostrich*) : در این روش در واقع هیچ عملی در مقابل بن بست انجام داده نمی شود. در صورتی که بن بست منجر به از کار افتادن سیستم شود(*Hang*) آنگاه سیستم به صورت دستی ری ست(*reset*) می شود.

□ جالب است که بدانید در اکثر سیستم عامل های امروزی مثل *unix* از همین روش چهارم استفاده می شود، چرا که در این سیستم ها بن بست به ندرت رخ می دهد(مثلا سالی یک بار) لذا ارزانتر آن است که به جای روش های پر هزینه پیشگیری، اجتناب و آشکار سازی کلا از این مشکل چشم پوشی کنیم.

ترکیب روش ها در اداره بن بست

در عمل هر یک از روش های اداره بن بست(پیشگیری،اجتناب،کشف) به تنهائی برای تمام انواع منابع مناسب نمی باشد، یک شیوه ترکیبی برای دسته های مختلف منابع مثلا می تواند به صورت زیر باشد.

- منابع داخلی سیستم مثل بلوک کنترل پردازش: پیشگیری از طریق ترتیب منابع
- منابع کار(گرداننده های دیسک ،نوار، چاپگر و...) : اجتناب از بن بست، چون حداکثر نیاز از قبل مشخص است
- حافظه اصلی: پیش گیری از طریق پس دادن می تواند انجام پذیرد، چرا که به راحتی می توان حافظه اصلی را از پردازه ها پس گرفت؛ زیرا به محض کمبود حافظه یکسری از پردزاه ها به حافظه جانبی منتقل می شوند.
- حافظه جابه جاپذیر (پشتیبان): تخصیص از پیش، میتواند انجام پذیرد چرا که حداکثر نیاز های ذخیره سازی از قبل می تواند مشخص باشد.
- منابع پردازش: از طریق اجتناب

پایان جلسه هشتم

جلسه نهم

مدیریت حافظه(قطعه بندی و صفحه بندی)**:**

☐ یکی از مولفه های سیستم عامل مدیریت حافظه است.

وظایف مدیر حافظه:

- اداره کردن سلسله مراتب حافظه
- جلوگیری از تداخل برنامه های موجود در حافظه(به خصوص در محیط های چند برنامه گی)
- مدیریت حافظه مجازی

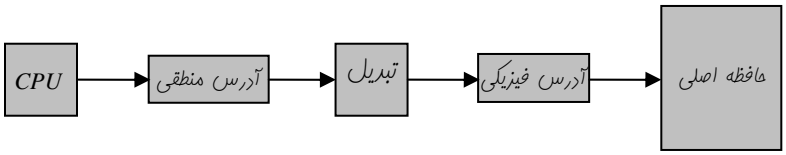
آدرس منطقی: آدرسی است که برنامه نویس در برنامه صادر میکند یا آدرسی که توسط *CPU* تولید میشود. مثلا آدرس منطقی 100 در کد *Mov Al*,[100]

آدرس فیزیکی:، آدرس مشاهده شده توسط واحد حافظه(یعنی آنچه که در رجیستر آدرس حافظه، بار میشود) را آدرس فیزیکی می نامند.

آدرس مجازی: در حالتی که پیوند آدرس های حافظه در زمان اجرا باشد، به آدرس منطقی، آدرس مجازی(*Virtual Addresss*) میگویند.

تبدیل آدرس منطقی به آدرس فیزیکی:

به عمل تبدیل آدرس منطقی به آدرس فیزیکی نگاشت آدرس(*maping*) گویند.



انواع انقیاد آدرس:

1- **زمان کامپایل:** اگر در موقع کامپایل معلوم باشد که برنامه در کجای حافظه قرار خواهد گرفت، در این صورت کد مطلق میتواند تولید شود، یعنی آدرس های ذکر شده در برنامه هنگام بارشدن و یا هنگام اجرا تغییر نخواهد کرد و تصویر آینه وار برنامه در دیسک عینا به به حافظه آورده شده و اجرا می گردد. مثلا آدرس 100 ذکر شده در برنامه همان آدرس 100 مطلق حافظه *RAM* می باشد.برنامه های *com* تحت سیستم عامل *DOS* اینگونه هستند.

2- **زمان بار کردن:** اگر در زمان کامپایل معلوم نباشد که برنامه در کجای حافظه قرار خواهد گرفت، آنگاه کامپایلر بایستی کد قابل جابه جائی (*Relocatable*)تولید کند.به عنوان مثال اگر در برنامه ای دستور *Mov Al* [100] را داشته باشیم و برنامه حاوی این دستور در زمان بار کردن در محل 300 حافظه قرار گیرد(شروع آدرس 300 باشد) در این صورت آدرس فیزیکی معادل آدرس 100 برابر 400 میشود.

3- **زمان اجرا:**اگر پردازش در حین زمان اجرایش بتواند در حافظه جابه جا شود، آنگاه پیوند دادن بایستی تا زمان اجرا به تاخیر انداخته شود. برای این حالت نیاز به سخت افزار خاصی وجود دارد.

☐ هم برای نگاشت زمان کامپایل و هم برای زمان نگاشت بار کردن مدیر حافظه ملزم به پشتیبانی سخت افزاری نیست ولی برای نگاشت زمان اجرا نیاز به پشتیبانی سخت افزاری است.

روش های مدیریت حافظه:

1- **تک برنامگی ساده:**

در این روش در هر لحظه فقط یک برنامه در حافظه اصلی قرار دارد، و برنامه ای که می بایست اجرا شود، نباید اندازه اش از حافظه اصلی بیشتر باشد اگر حافظه *RAM* به اندازه کافی در دسترس نباشد، برنامه اجرا نمی شود. سیستم عامل *DOS* اولیه این گونه بوده است.

طرح حافظه در این سیستم عامل اولیه به این شکل بوه است

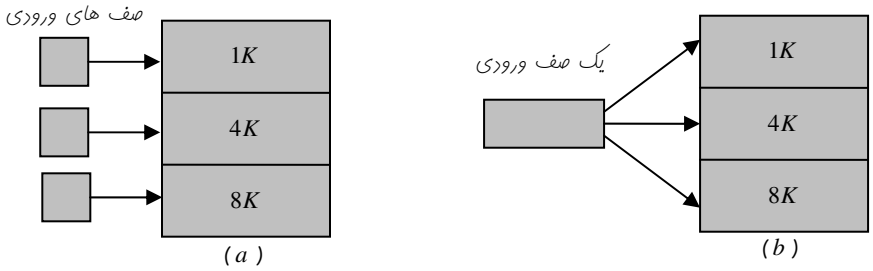
2- یک برنامه گی با سیستم (overlay)

در این سیستم مدیریت، پردازش می تواند بزرگتر از حافظه اصلی باشد. در شیوه overlay(جایگذاشت) برنامه به بخش های مختلفی تقسیم شده و تنها آن داده ها و دستورالعمل هائی را در حافظه قرار می دهیم که در هر زمان مفروض مورد نیاز هستند و بقیه بخش ها در دیسک باقی می مانند. هنگامی که به بخش دیگری از آن برنامه نیاز داریم ، قسمتی که مورد نیاز نیست از حافظه خارج شده و بخش مورد نیاز به حافظه آورده می شود .

- overlay به حمایت سخت افزاری ویژه ای نیاز ندارد. این تکنیک را خود برنامه نویس می بایست در برنامه پیاده سازی کند .

3- چند برنامگی با بخش بندی ثابت حافظه:

ساده ترین روش چند برنامه گی این است که حافظه را به N قسمت تقسیم کنیم، اندازه هر قسمت می تواند با بخش های دیگر متفاوت باشد.این کار می تواند در هنگام شروع کار سیستم توسط سیستم عامل یا به صورت دستی توسط اپراتور انجام شود. وقتی یک کار وارد می شود در یک صف ورودی قرار می گیرد تا در کوچکترین بخش که مناسب آن است قرار داده شود. البته ممکن است آن بخش دقیقا هم اندازه برنامه نبوده و بدین ترتیب مقداری از فضای آن از بین برود.دراین روش می توان برای هر پارتیشن از یک صف مجزا استفاده کرد(شکل a) و یا اینکه فقط یک صف برای تمام پارتیشن ها داشت(شکل b).



4- چند برنامگی با جابه جائی(Swapping)–مبادله

در این روش در هر لحظه میتوان چندین پردازه داشت، ولی اگر پردازه ای می خواهد به حافظه اصلی بیاید، و حافظه خالی نباشد، به جای یکی از پردازه های موجود در حافظه اصلی قرار می گیرد، و پردازه ای که در حافظه اصلی بود به دیسک منتقل میشود. اشکالی که در این روش وجود دارد این است که به دلیل نقل انتقال پردازه مابین حافظه اصلی و دیسک زمانی طول می کشد. $(s+r+b_{tt})=$ زمان جابه جائی پردازه ها

$$\text{کارائی } CPU = \frac{\text{زمان اجرای پردازه}}{\text{زمان جابه جائی+زمان اجرای}}$$

$$\text{کل زمان طی شده}$$

5- چند برنامگی به صورت تخصیص همجوار:

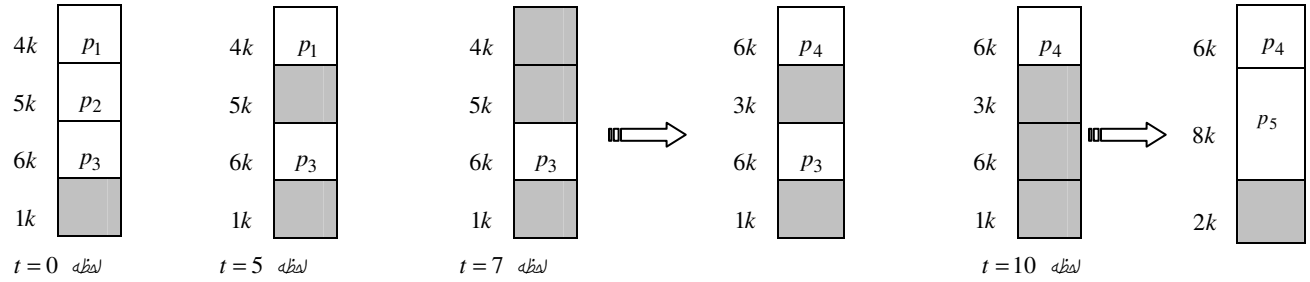
در این روش حافظه از قبل به اندازه های ثابت تقسیم نمی شود بلکه پردازه هائی که می بایست به حافظه آیند مطابق الگوریتم هائی یکی از فضا های آزاد حافظه را پیدا کرده و به طور کامل در آن قسمت قرار می گیرد.

مثال. فرض کنید اندازه حافظه ای $16k$ باشد اگر کار های زیر با توجه به الگوریتم FCFS توسط زمان بند بلند مدت جهت تبدیل شدن به پردازه ها انتخاب شوند و در حین اجرای پردازه ها از زمانبندی RR با کوانتوم زمانی 1 استفاده شود. وضعیت حافظه را بعد از هر Load شدن برنامه جدید به حافظه مشخص نموده و زمان Load شدن پردازه ها را معین کنید(پردازه وارد شده به انتهای صف می رود).

job	مقدار حافظه مورد نیاز	زمان اجرا
j_1	$4k$	3
j_2	$5k$	2
j_3	$6k$	4
j_4	$6k$	3
j_5	$8k$	3

ابتدا برنامه های p_3, p_2, p_1 در حافظه بار شده و شروع به اجرا میشوند. توجه کنید به علت نبود حافظه کافی

پردازه های p_5, p_4 در حافظه بار نمی شوند.



p_1	p_2	p_3	p_1	p_2	p_3	p_1	p_3	p_4	p_3	p_4	p_5	p_4	p_5	p_5
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

پارگی:

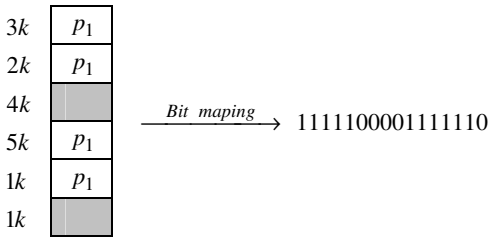
به حفره های حافظه که ظرفیت مجموع آنها زیاد بوده ولی چون از همدیگر فاصله دارند قابل استفاده نیستند را پارگی خارجی گوئیم که یک راه برای از بین بردن آنها فشرده سازی می باشد(کاری می کنیم که فضا های آزاد در کنار هم قرار گیرند یعنی برنامه ها را جابه جا کنیم)، فشرده سازی عملی زمانبر هست و از طرفی کد هائی را می توان جابه جا کرد که جابه جاپذیر(***Relocatable***) باشند.

□ اگر زمانی پردازه ای مانند j_4 بخواهد وارد حافظه اصلی شود یک فضای $7k$ خالی در ابتدای حافظه و یک فضای خالی $6.5k$ در محل دیگر حافظه باشد، در کدام محل قرار گیرد؟ جواب دادن به سئوال هائی از این قبیل به بحث الگوریتم های انتخاب بستگی دارد.

روش های مدیریت فضای آزاد

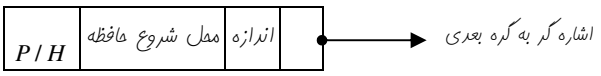
الف. روش نگاشت بیتی(*Bit maps*)

در این روش یک واحد مناسب در نظر گرفته می شود، و متناظر با این واحد یک بیت در نظر گرفته میشود، بنابراین با شروع از ابتدای حافظه هر واحدی از حافظه که پر باشد بیت متناظر آن 1 و اگر خالی باشد بیت متناظر آن صفر در نظر گرفته می شود. مشکل این روش این است که اگر پردازه به k واحد نیاز داشته باشد، می بایست در رشته بیتی حافظه به دنبال k صفر کنار هم باشیم.به عنوان مثال اگر واحد انتخابی $1k$ باشد. شکل زیر این موضوع را روشن می کند.



ب.روش لیست پیوندی(*Linked list*)

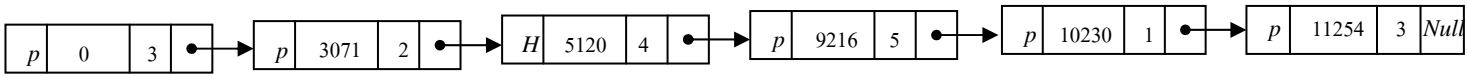
در روش لیست پیوندی با شروع از ابتدای حافظه به ازای هر پردازه یا فضای خالی یک گره ای با ساختار زیر در نظر گرفته می شود.



P : پردازه

H : فضای آزاد

مثال قبلی به روش لیست پیوندی



الگوریتم های انتخاب جا:

1- اولین مناسب(*frist fit*)

وقتی پردازه ای می خواهد به حافظه load شود ابتدای لیست فضای آزاد را نگاه کرده و اولین فضای آزادی که اندازه اش بزرگتر یا مساوی اندازه پردزاه باشد انتخاب شده و پردازه در آن محل قرار می گیرد.

☐ *مشکل تراکم پردازه ها در ابتدای حافظه است که روش بعدی سعی می کند این مشکل را برطرف کند.*

2- مناسب بعدی(*Next fit*)

این روش مانند frist fit است با این تفاوت که جستجو از محلی در لیست آغاز می شود که آخرین بار تخصیص از آن محل صورت گرفته است. بدین ترتیب یکنواختی توزیع برنامه ها در سطح حافظه نسبت به روش قبلی بیشتر خواهد شد.

3- بهترین مناسب(*Best fit*)

در این روش کل لیست فضای آزاد جستجو شده و کوچکترین حفره که به اندازه کافی بزرگ است به پردزاش تخصیص داد می شود. این روش باعث می شود که کوچکترین حفره بر اثر تخصیص باقی بماند. با این روش فضا های بزرگتر برای تقاضا های بیشتر حفظ می شوند. از آنجا که تمام لیست بلاک های آزاد باید بررسی شود، این تکنیک قدری زمانبر است.

پایان جلسه نهم

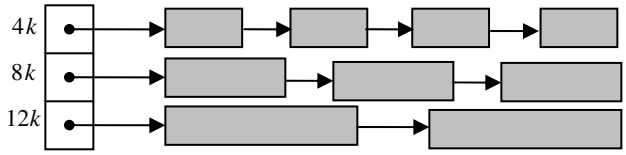
جلسه دهم:

4- بدترین مناسب($worst-fit$):

در این الگوریتم بزرگترین حفره انتخاب شده و پردازه در آن قرار میگیرد. دلیل انتخاب بزرگترین حفره این است که از فضای باقی مانده دیگر پردازه ها می توانند استفاده کنند.ایراد این تکنیک این است که امکان دارد، تقاضاهای که ناحیه بزرگی می خواهند، دیگر نتوانند برآورده شوند چرا که بلاک های بزرگ زودتر تخصیص یافته و کوچک می شوند.

5- سریعترین مناسب($Quick\ Fit$):

در این الگوریتم لیستی از اندازه پردازه های متداول تهیه می شود و آرایه ای با n خانه در نظر گرفته می شود که هر خانه این آرایه شامل یک اشاره گر به ابتدای لیست یک فضای خالی به اندازه متداول است به عنوان مثال فضا های متداول می توانند $2k$, $4k$, $8k$, $12k$ و ... باشند که برای هر کدام یک خانه آرایه در نظر گرفته می شود.عیب این روش این است که اگر پروسسی خاتمه یابد باید فضای آزاد شده آن به لیست مناسب اضافه شود که این کار زمانبر می باشد.



6- الگوریتم رفاقتی($Buddy$):

در این روش حفره ها(فضا های خالی) به صورت توان های 2 در نظر گرفته می شود.به عنوان مثال حفره هائی به اندازه $1k$, $2k$, $4k$, $8k$, $16k$, $32K$... و برای هر گروه یک لیست جداگانه در نظر گرفته می شود.بدین ترتیب جهت تخصیص یک بلاک تنها باید بلاک مورد نظر را از لیست مناسب خارج کرد. پس از تخصیص اگر فضای باقی مانده آن بلاک، توانی از 2 باشد در لیست مربوطه اش قرار می گیرد و در غیر این صورت به چندین بخش که اندازه هر کدام توانی از 2 می باشد تقسیم میشود. از طرف دیگر در این روش بلوک های کنار هم می توانند باهم ترکیب شده و بخش بزرگتری را پدید بیاورند.

مثال. با قسمت هائی از حافظه به اندازه هائی $100\,k$, $500\,k$, $200\,k$, $300\,k$, $600\,k$ هر یک از روش های اولین جای مناسب، بهترین جای مناسب و بدترین جای مناسب پردازه هائی با اندازه $212k$, 417, $112k$, $426k$ را چگونه در حافظه قرار می دهند و کدام روش از حافظه به طور بهینه استفاده می کند. ترتیب ورود پردازه ها را یک بار از راست به چپ و یک بار از چپ به راست بگیرید.

$H: 100\,k\ ,\ 500\,k\ ,\ 200\,k\ ,\ 300\ ,\ 600\,k$
$P: 212\,k\ ,\ 417\ ,\ 112\,k\ ,\ 426\,k$

الف. اولین مناسب: در حالت از چپ به راست

$100k\ ,\ 176k\ ,\ 200k\ ,\ 300k\ ,\ 183k\ ,$ لیست فضای آزاد، ترتیب پردازه ها از چپ به راست
$100k\ ,\ 74k\ ,\ 88k\ ,\ 88k\ ,\ 183k$ لیست فضای آزاد، ترتیب پردازه ها از راست به چپ

پردازه 426 باید منتظر بماند

ب. بهترین مناسب

$100\ k\ ,\ 83\ k\ ,\ 88\ k\ ,\ 88\ k\ ,\ 174\ k\ ,$ لیست فضای آزاد، ترتیب پردازه ها از چپ به راست
$100\ k\ ,\ 74\ k\ ,\ 88\ k\ ,\ 88\ k\ ,\ 183\ k$ لیست فضای آزاد، ترتیب پردازه ها از راست به چپ

ب.بدترین مناسب

$100\ k\ ,\ 388\ k\ ,\ 200\ k\ ,\ 88\ k\ ,\ 174\ k\ ,$ لیست فضای آزاد، ترتیب پردازه ها از چپ به راست
$100\ k\ ,\ 83\ k\ ,\ 200\ k\ ,\ 188\ k\ ,\ 388\ k$ لیست فضای آزاد، ترتیب پردازه ها از راست به چپ

روش بهترین مناسب از حافظه به صورت بهینه استفاده می کند. زیرا برای تمام پردازه ها فضای لازم را پیدا می کند و پارگی خارجی در آن حداقل است.

مثال. در زیر بلوک های خالی حافظه به ترتیب از چپ به راست نشان داده شده اند، اگر درخواست های جدیدی برای چهار بلوک به اندازه $20k$, $30k$, $20k$, $35k$ به ترتیب از راست به چپ ذکر شده داده شود و از روش $Next\ Fit$ استفاده شود و تخصیص از اول حافظه شروع شود، وضعیت حافظه را بعد از این تخصیص ها مشخص کنید.

شروع $\rightarrow 40k\ ,\ 25k\ ,\ 45k\ ,\ 50k\ ,\ 60k\ ,\ 40k$

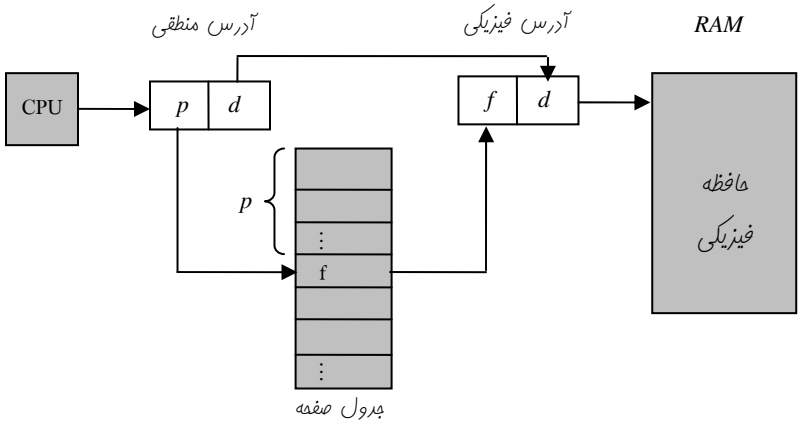
جواب: $20\,k\ ,\ 25\,k\ ,\ 15\,k\ ,\ 30\,k\ ,\ 25\,k\ ,\ 40\,k$

صفحه بندی(Paging):

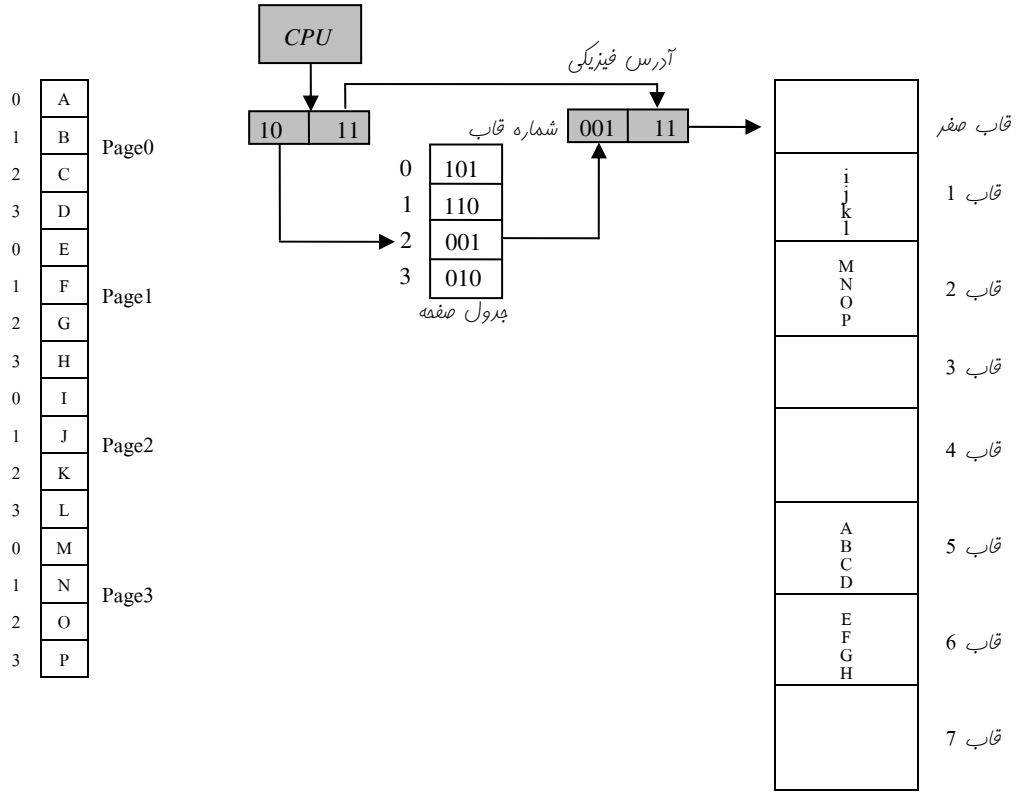
در روش صفحه بندی برنامه کاربر(فضای آدرس منطقی) به بخش هائی به اندازه ثابت به نام صفحه(Page) تقسیم می شود و حافظه فیزیکی به قسمت هائی به نام قاب(Frame) تقسیم می شود که اندازه هر فریم برابر اندازه Page می باشد. صفحه بندی این امکان را می دهد که قسمت های یک برنامه (Page های هر برنامه) در حافظه پراکنده باشند(لازم نیست مجاور هم باشند)

نحوه تبدیل آدرس منطقی به آدرس فیزیکی:

در این تکنیک هر آدرس تولید شده توسط CPU(یعنی آدرس منطقی) از دو بخش شماره صفحه(p) و افست صفحه(d) تشکیل شده است. شماره صفحه به عنوان اندیس جدول صفحه(page table) استفاده می گردد. جدول صفحه شامل آدرس مبنای هر صفحه در آدرس فیزیکی RAM است. این آدرس مبنا با آدرس افست منطقی ترکیب شده و آدرس فیزیکی نهائی را تشکیل می دهد. شکل زیر این موضوع را نشان می دهد.



مثال. فرض کنید تعداد صفحات $2^2 = 4$ عدد و تعداد قاب ها $2^3 = 8$ عدد باشد.



□ به هر کدام از سطر های جدول صفحه یک مدخل یا Entry گویند. که هر مدخل شامل شماره یک فریم و اطلاعات دیگری از قبیل بیت معتبر(نامعتبر)، بیت Read، بیت write و بیت های دیگر می باشد.

$\lceil Log_2^{\text{تعداد صفحات}} \rceil =$ تعداد بیت های مورد نیاز برای شماره جدول صفحه(p)

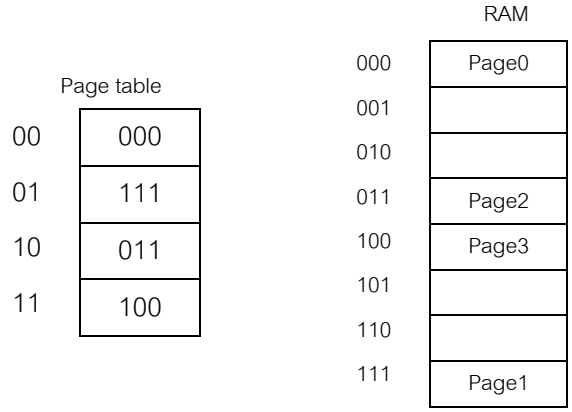
حافظه منطقی ≥ حافظه فیزیکی ⇒ تعداد *page* ها ≥ تعداد فریم

$\lceil Log_2^{\text{اندازه صفحات}} \rceil =$ تعداد بیت های مورد نیاز برای افست(d)

□در صفحه بندی اگر بیت *valid* صفر باشد بدین معناست که آدرس صفحه ذکر شده نامعتبر می باشد یا چنین صفحه ای وجود ندارد.

جلسه یازدهم و دوازدهم

مثال. اگر اندازه هر page ، 1k باشد تعداد خطوط آدرس منطقی را با مشخص کردن d وp و همینطور تعداد خطوط آدرس فیزیکی را با مشخص کردنd وf بدست آورید. آدرس منطقی و آدرس فیزیکی صفحه شماره 1 و خانه شماره 20 را مشخص کنید.



حل.

تعداد بیت های d= $\left|\log_2^{page\ size}\right| = \left|\log_2^{1024}\right| = 12$

d	F
10 بیت	3 بیت

d	p
10 بیت	2 بیت

با توجه به شکل چهار صفحه داریم که دو بیت برای آدرس دهی آن کافی است پس برای آدرس دهی منطقی 12 بیت لازم است و

با توجه به شکل هشت قاب داریم که سه بیت برای آدرس دهی آن کافی است پس برای آدرس دهی فیزیکی 13 بیت لازم است

آدرس منطقی صفحه شماره 1و خانه شماره 20

0000010100	01

آدرس فیزیکی صفحه شماره 1و خانه شماره 20

0000010100	111

☐ چون جدول صفحه در حافظه اصلی نگهداری میشود بنابراین هر مراجعه به حافظه تبدیل به دو مراجعه می شود، یک مراجعه به جدول صفحه جهت بدست آوردن شماره frame و مراجعه دیگر جهت بدست آوردن داده اصلی پس $t_{acc} = 2t_m$ که t_m زمان یک مراجعه می باشد.

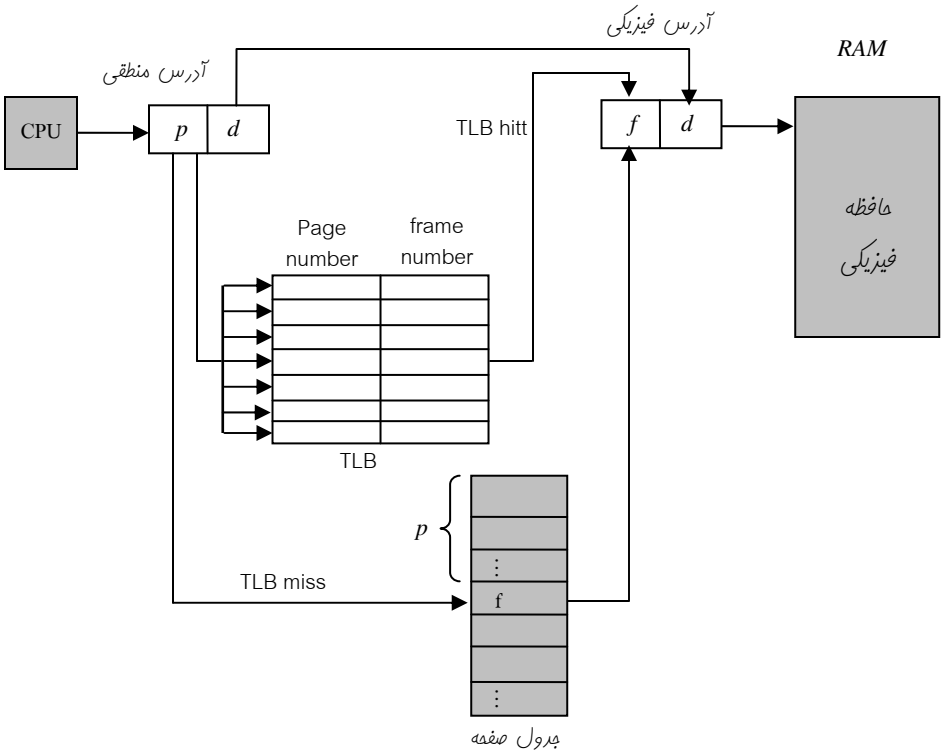
☐ چون این روش زمانبر است معمولا جهت نگهداری جدول صفحه از بافر های دم دستی استفاده می شود (*TLB*)که از نوع حافظه شرکت پذیر می باشند. که به طور موازی می توان در آنها عمل جستجو را انجام داد.

روش های استفاده از *TLB* جهت نگهداری جدول صفحه

1- نگهداری کامل جدول صفحه در *TLB*

اشکال: اندازه جدول صفحه خیلی بزرگتر از *TLB* می باشد(اندازه *TLB* محدود می باشد زیرا گران است)

2- از جدول صفحه که در حافظه اصلی است به همراه *TLB* استفاده می شود که در این صورت جهت بدست آوردن شماره frame متناظر با یک شماره صفحه ابتدا عمل جستجو در *TLB* انجام میشود، اگر شماره صفحه مورد نظر در *TLB* یافت نشود به جدول صفحه در حافظه اصلی مراجعه میشود و بعد از این عمل *TLB* بروز(update) می شود.شکل صفحه بعد این موضوع را نشان میدهد.



مثال.

اگر جستجوی *TLB* به اندازه 20 نانو ثانیه طول بکشد و دستیابی به حافظه اصلی نیز 100 نانو ثانیه زمان بخواهد، آنگاه زمان دستیابی به حافظه را موقعی که شماره قاب در *TLB*پیدا شود و هنگامی که در *TLB*پیدا نشود را بدست آورید.

حل: زمان دستیابی اگر در *TLB* پیدا شود. $20+100=120 \quad ns$

زمان دستیابی اگر در *TLB* پیدا نشود. $20+100+100=220 \quad ns$

تذکر: جدول *TLB*در واحد مدیریت حافظه یا *MMU* (*memory Management unit*)قرار دارد، خود *MMU* نیز در *CPU* قرار دارد.

احتمال وجود شماره صفحه درh:TLB

عدم وجود(عدم موفقیت):1-h

زمان دستیابی به حافظه اصلی: t_m

$t_{acc} = h\,t_{TLB} + (1-h)(t_{TLB} + t_m) + t_m$

مثال. یک سیستم صفحه بندی را در نظر بگیرید، جدول صفحه در حافظه اصلی است اگر زمان دستیابی به حافظه اصلی برابر60 نانو ثانیه باشد و احتمال وجود شماره صفحه در *TLB*، 0.75 باشد و زمان دستیابی به *TLB*،5 نانو ثانیه باشد ،نسبت بهبود آدرس بر اثر *TLB*، در مقاسه با هنگامی که از *TLB*، استفادره نمی شود چقدر است؟

h=0.75

1-h=0.25

t_m =60 ns

t_{TLB} =5 ns

زمان دستیابی به داده اصلی بدون استفاده از *TLB* $t_{acc} = 2t_m = 120 \;\; ns$

زمان دستیابی به داده اصلی با استفاده از *TLB* $t_{acc} = 0.75(5) + (0.25)(5+60) + 60 = 80 \;\; ns$

نسبت بهبود آدرس $= \frac{120}{80}$

مثال. در یک سیستم حافظه صفحه بندی با یک جدول صفحه حاوی64 مدخل 11 بیتی (شامل یک بیت اعتبار/عدم اعتبار) و صفحه هایی به اندازه هر یک512 بایت یک آدرس منطقی و یک آدرس فیزیکی چند بیت است؟

حل.چون جدول صفحه 64 مدخل و هر صفحه 512 بایت است بنابراین $\text{اندازه حافظه منطقی} = 64 \times 512 = 2^6 \times 2^{15} = 2^{15}$

پس آدرس منطقی 15 بیتی است، از طرفی چون هر مدخل جدول صفحه برای آدرس دهی 10 بیتی است(یک بیت برای عملیات کنترلی است) و هرآدرس موجود در هر مدخل(سطر) جدول صفحه به یک page با اندازه 512 بایت اشاره می کند پس $\text{اندازه حافظه فیزیکی} = 2^{10} \times 512 = 2^{19}$

لذا آدرس فیزیکی 19 بیتی است.

PTBR : همواره حاوی آدرس شروع جدول صفحه پردازه در حال اجراست، هنگام ***context swiching*** ، ***PTRB*** برابر آدرس شروع جدول صفحه پردازه جدید میشود.

بعضی از مدیریت حافظه بایستی به صورت سخت افزاری باشد(پشتیبانی شود)

مثال. فرض کنید آدرس منطقی32 بیتی و اندازه هر صفحه(قاب) نیز 1kB باشد در این صورت مطلوب است تعیین

الف. تعداد بیت های p و d

ب. اندازه جدول صفحه در صورتی که هر مدخل (Entitiy) جدول صفحه 8 بایت باشد.

حل. با توجه به این که اندازه هر صفحه یا قاب 1kB می باشد پس($1k = 2^{10}$) 10 بیت برای قسمت آفست(d) لازم است. و با توجه به این که آدرس منطقی32 بیتی می باشد، پس برا ی قسمت p ، 22 بیت باقی می ماند.

p	d
22 بیت	10 بیت

برای حل قسمت ب ابتدا بایستی تعداد مدخل های جدول صفحه را بدست آوریم

$\text{تعداد مدخل های جدول صفحه} = 2^{22} = 4M$

$\text{اندازه جدول صفحه} = 4M \times 8 = 32M \ \ byte$

☐ چون اندازه جدول صفحه خیلی بزرگ می شود از روش صفحه بندی چند سطحی استفاده می شود به عبارتی برای جداول صفحه نیز جدول صفحه ایجاد می کنیم.

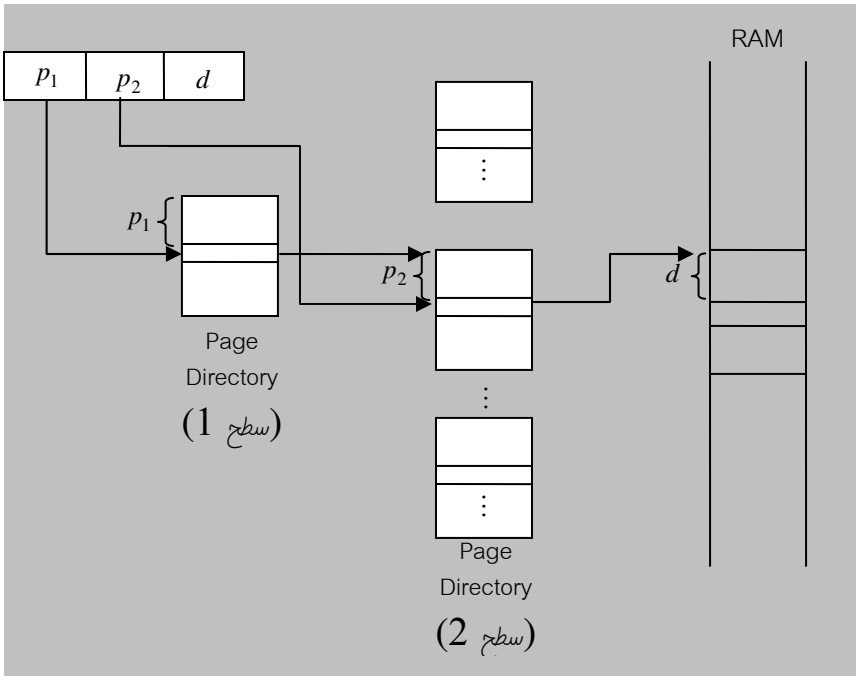
صفحه بندی چند سطحی

اغلب کامپیوترهای امروزی دارای فضای آدرس دهی منطقی بسیار بزرگی هستند(2^{32} تا 2^{64} خانه آدرس) در چنین سیستم هائی خود جدول بسیار بزرگ خواهد بود، برای رفع این مشکل می توان از تکنیک صفحه بندی دو سطحی استفاده کرد به گونه ای که در آن جدول صفحه، خود صفحه بندی شده باشد در واقع در این روش جدول صفحه به قطعات کوچک تقسیم شده و دیگر لازم نیست تمامی جداول صفحه درRAM نگهداری شوند به عبارت دیگر جداولی که به آنها فعلا نیاز نداریم به حافظه آورده نمی شوند

مثلا مدل آدرس دهی در اکثر پردازنده های32 بیتی به صورت زیر است

← Page number →		← offset →
P_1	P_2	d
10 بیت	10 بیت	12 بیت

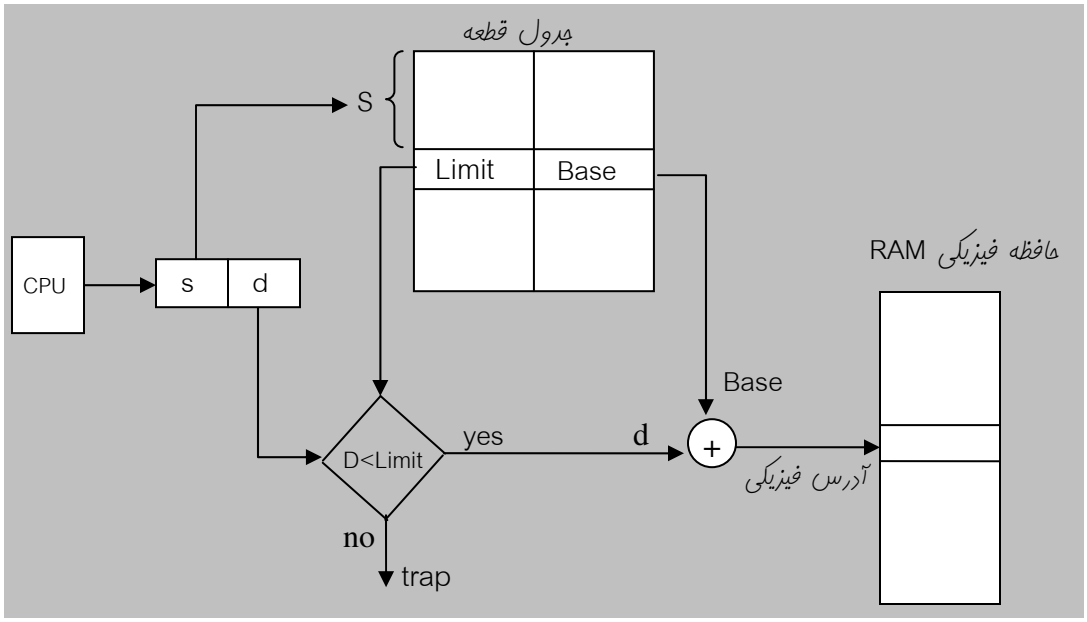
چون آفست 12 بیتی است پس اندازه هر صفحه $2^{12} = 4k$ بوده و حداکثر 2^{20} صفحه وجود خواهد داشت($2^{32} \div 2^{12} = 2^{20}$).هر مدخل p_1 آدرس شروع یک جدول صفحه را در سطح 2 می دهد و هر مدخل p_2 آدرس شروع یک page (یک frame) را می دهد و بیت های d یک آدرس را در داخلpage (یا frame) مشخص می کند. شکل صفحه بعد نحوه این آدرس دهی را نشان می دهد.



بدین ترتیب هر پردازش یک *page Directory* با $(2^{10} = 1k)$ هزار ورودی دارد که هر ورودی آن می تواند به یک *page Table* اشاره کند که آن نیز $(2^{10} = 1k)$ هزار ورودی دیگر دارد. برای هر پردازش همواره *page Directory* در حافظه قرار می گیرد ولی *page Table* های سطح2، به تعداد لازم در حافظه قرار می گیرند.

قطعه بندی(segmentation)

حافظه اصلی یا فیزیکی به صورت یک آرایه خطی از بایت ها می باشد به عبارتی هر خانه در حافظه یک آدرس دارد، در روش صفحه بندی پردازه ها بر اساس محدودیت فیزیکی یعنی اندازه هر صفحه تقسیم می شوند ولی در قطعه بندی خود کاربر برنامه نویس می تواند برنامه اش را به صورت منطقی تقسیم کند که به هر یک از این قسمت ها یک segment گویند هر قطعه یک آدرس شروع و یک طول دارد، آدرسی که کاربر در سطح منطقی می دهد شامل دو جزء است. یکی شماره قطعه و دیگری فاصله(آفست) خانه مورد نظر از اول آن قطعه است، آدرس دو بعدی استفاده شده توسط کاربر در سطح منطقی می بایست توسط یک نگاشت به آدرس یک بعدی فیزیکی تبدیل شود. این نگاشت به وسیله جدول قطعه انجام می پذیرد، جدول قطعه از دو ستون اصلی تشکیل شده است، یکی آدرس پایه قطعه(Base) و دیگری طول یا حد قطعه(Limit). قسمت Base حاوی آدرس فیزیکی در RAM می باشد که قطعه از آنجا شروع می شود، شکل زیر نحوه تبدیل آدرس منطقی را به آدرس فیزیکی در این سیستم نشان میدهد.



آدرس منطقی از دو جزء شماره قطعه(s) و آفست درون آن قطعه(d) تشکیل یافته است.

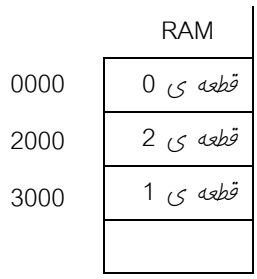
- یکی از مزایای قطعه بندی اشتراک است.
- در صفحه بندی اندازه صفحات برابر است ولی در قطعه بندی لزومی ندارد که اندازه صفحات یکی باشد

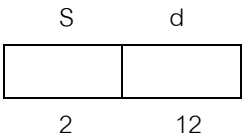
مثال. فرض کنید در یک شبکه کامپیوتری چندین کاربر به صورت همزمان نیاز به اجرای برنامه word دارند به جای این که هر برنامه word که برای همه مشترک است برای هر کاربر به صورت مجزا در حافظه load شود، فقط یک بار در حافظهload شده و تمامی کاربران به صورت اشتراکی از آن استفاده می کنند و هر کاربر قطعه خاص خود را دارد.

مثال. فرض کنید پردازه ای دارای سه قطعه می باشد به طوریکه قطعه اول2k ، قطعه دوم3k و قطعه سوم4k می باشد مطلوب است

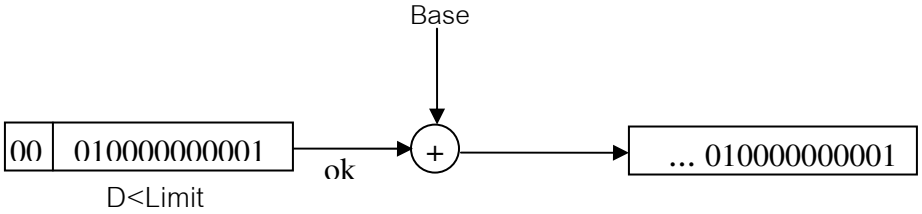
الف. تعیین ورودیهای جدول قطعه

	base	Limit	V
00	0000	2×2^{10}	1
01	3000	3×2^{10}	1
10	2000	4×2^{10}	1
11			0





ب. اگر برنامه نویس در قطعه صفر دستور push **با** آدرس 1025 **را** داشته باشد آدرس فیزیکی متناظر با این آدرس را بیابید.



نکته: اگر برنامه نویس دستور pop خانه 2048 را در قطعه ی صفر بدهد چون آدرس صادر شده در قطعه صفر وجود ندارد بنابر این وقفه رخ میدهد

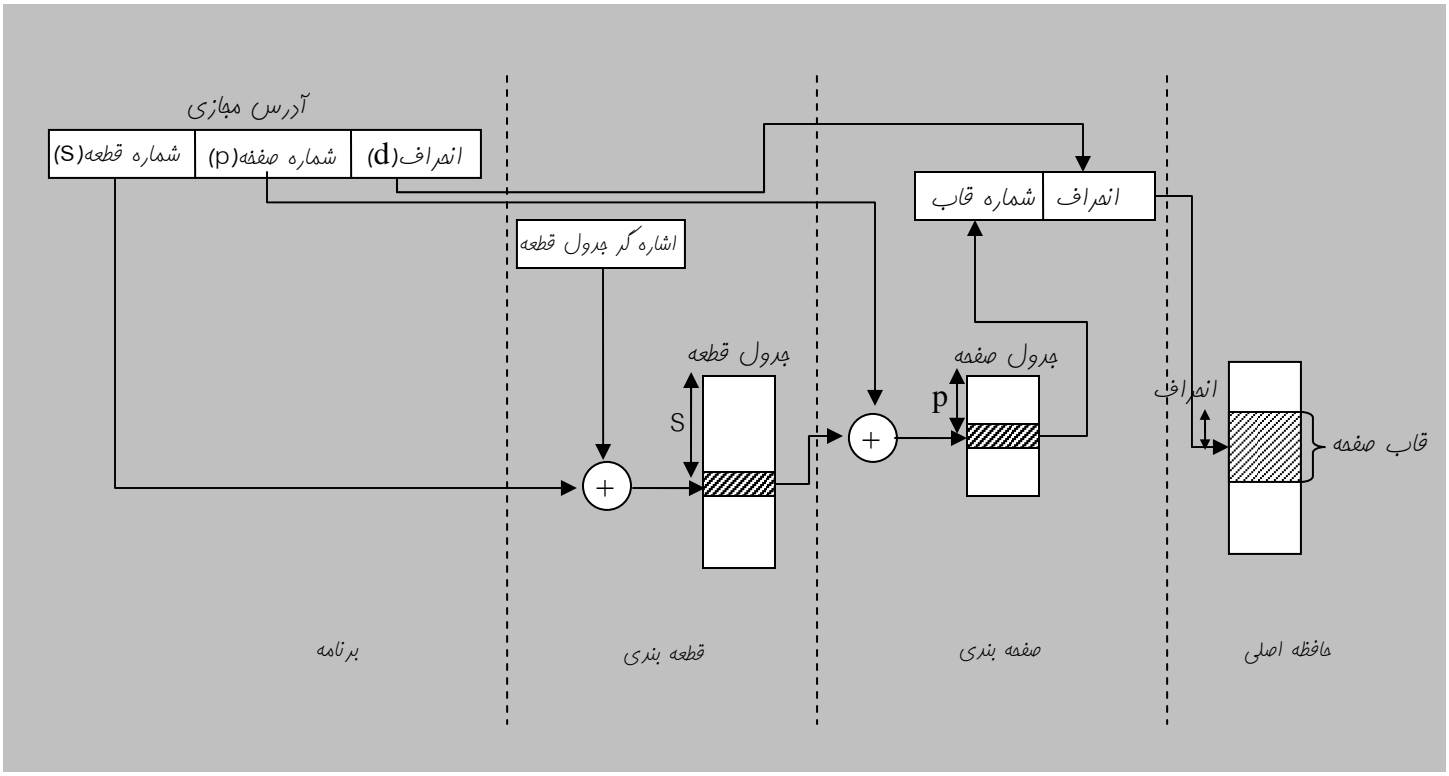
ج. اگر برنامه نویس دستور $Mov\ \ AX,[100]$ را صادر کند، آدرس موثر مطلق(فیزیکی) را بیابید

ترکیب قطعه بندی و صفحه بندی

هم صفحه بندی و هم قطعه بندی نقاط قوت مخصوص به خود را دارند برای ترکیب نقاط قوت هر دو، این دو روش را ترکیب کرده و با هم استفاده می کنیم، در یک سیستم ترکیبی صفحه بندی/ قطعه بندی فضای آدرس کاربر تحت نظر برنامه ساز به تعدادی قطعه تقسیم میشود، هر قطعه به نوبه خود به تعدادی صفحه به اندازه ثابت تقسیم می گردد. به ازای هر فرایند یک جدول قطعه و به ازای هر قطعه یک جدول صفحه ایجاد می گردد، تعداد درایه های هر جدول صفحه بستگی به اندازه قطعه مربوطه دارد. همانند صفحه بندی آخرین صفحه هر قطعه معمولا پر نمی شود و به ازای هر قطعه به طور میانگین به اندازه نصف صفحه تکه تکه شدن داخلی داریم. از دید برنامه ساز، آدرس

منطقی همچنان شامل شماره قطعه و انحراف در قطعه است. از دید سیستم این انحراف در قطعه، به صورت یک شماره صفحه و انحراف در صفحه دیده می شود.

هنگامی که فرایندی در حال اجراست یک ثبات آدرس شروع جدول قطعه آن فرایند را نگه می دارد. پردازنده از شماره قطعه ای که در آدرس منطقی است به عنوان شاخص به جدول قطعه استفاده کرده و جدول صفحه را برای قطعه مذبور پیدا می کند، سپس بخش شماره صفحه آدرس منطقی به عنوان شاخص جدول صفحه به کار رفته و شماره قاب مربوطه بدست می آید. این شماره قاب با بخش انحراف آدرس منطقی ترکیب شده و آدرس فیزیکی مورد نظر نتیجه میشود. شکل زیر نحوه ترجمه آدرس در یک سیستم قطعه بندی/صفحه بندی را نشان می دهد



برای مثال سیستم با آدرس منطقی 34 بیتی را در نظر بگیرید که در آن شماره قطعه 18 بیتی و انحراف16 بیتی است، به کار گیری مدیریت حافظه قطعه بندی در این سیستم به خاطر اندازه بزرگ قطعه که می تواند تا 2^{16} باشد مشکلاتی را که در فوق متذکر شدیم در بر می گیرد. بنابراین سیستم از مدیریت حافظه قطعه بندی/صفحه بندی استفاده می نماید. نگرش این مدیریت در این سیستم بدین گونه است که انحراف قطعه به دو بخش6 بیتی شماره صفحه و 10 بیتی انحراف صفحه تجزیه می شود. جدول صفحه برای هر قطعه می تواند حداکثر از 2^6 درایه بر خوردار باشد و هر فرایند حداکثر می تواند تا 2^{18} قطعه را دربر گیرد .

انحراف از صفحه(10 بیت	شماره صفحه(6 بیت)	شماره قطعه(18 بیت

مهمترین مزیت صفحه بندی:

پارگی خارجی ندارد

در قطعه بندی پارگی خارجی داریم زیرا اندازه قطعات مساوی نیست و قطعه ای بلااستفاده ای وجود دارد که پراکنده بودن و همجوار نیستند بنابراین نمی توان از آنها برای یک قطعه استفاده کرد.

مهمترین مزیت قطعه بندی

اشتراک

اشتراک در صفحه بندی نسبت به قطعه بندی ناچیز است و یا اصلا وجود ندارد.

جدول زیر مقایسه ای بین روش های مدیریت حافظه را نشان می دهد.

	چند برنامگی	جهت اجرا کل برنامه در حافظه می باشد یا نه	آیا کل برنامه در حافظه اصلی کنار هم قرار می گیرد
تک برنامه گی ساده	یک	کل برنامه	کنار هم
تک برنامه گی با overlay	یک	لزومی ندارد	کنار هم
چندبرنامه گی با swapping	چند	کل برنامه	کنار هم
چند برنامه گی به صورت همجوار	چند	کل برنامه	کنار هم
چند برنامه گی به صورت بخش بندی	چند	کل برنامه	کنار هم
صفحه بندی	چند	کل برنامه	لزومی ندارد
قطعه بندی	چند	کل برنامه	لزومی ندارد

جلسه دوازدهم و چهاردهم

حافظه مجازی:

تکنیکی است که اجازه می دهد بدون این که کل برنامه در حافظه اصلی قرار گیرد اجرا شود.**مزیت مهم** این روش آن است که اجازه می دهد برنامه بزرگتر از حافظه اصلی باشد.علاوه بر این در این روش کاربر حافظه را به صورت آرایه ای فوق العاده بزرگ و یکنواخت می بیند.به عبارتی کاربر(برنامه نویس) خود را درگیر حافظه فیزیکی نمی کند و فرض می کند که بی نهایت حافظه در اختیار دارد و برنامه نویسی ساده می شود.**مزیت دیگر** این است که چون فقط صفحات مورد نیاز از دیسک به حافظه اصلی منتقل می شوند بنابراین زمان کمتری صرف عملI/O می شود از این رو زمان پردازش نیز سریعتر می شود.

جهت پیاده سازی حافظه مجازی دو روش وجود دارد.

- ترکیبی از صفحه بندی و swapping
- ترکیبی از قطعه بندی و swapping

به عبارتی حافظه مجازی به صورت صفحه بندی نیازی پیاده سازی می شود. در این روش مطابق اصل محلی بودن مراجعه به حافظه صفحه یا صفحاتی را که مورد نیاز باشند به حافظه اصلی آورده می شوند و هر وقت به صفحه ای نیاز باشد ولی در حافظه اصلی نباشد آن صفحه مطابق روشی که بعدا خواهیم گفت به حافظه اصلیLoad می شود.

حافظه فیزیکی به یک سری قسمت های مساوی به نام Frame یا قاب تقسیم می شود. و برنامه کاربر نیز به قسمت هائی به اندازه هر Frame به نام page یا صفحه تقسیم می شود.

☐ اگر در ابتدا هیچ صفحه ای در حافظه اصلی نباشد و به محض نیاز، به حافظه اصلی آورده شود به این روش **صفحه بندی نیازی محض** گویند. در این روش هرگز صفحه ای تا وقتی که مورد نیاز نباشد وارد حافظه نمی شود.

کارائی صفحه بندی نیازی

اگر t_m زمان دستیابی به حافظه، p احتمال وقوع خطای صفحه(احتمال این که صفحه در حافظه اصلی نباشد) و t_f زمان لازم برای سرویس دهی به خطای صفحه باشد، آنگاه زمان دسترسی موثر (t_{eff})از فرمول زیر محاسبه می شود.

$$t_{eff} = (1-p)t_m + p \times t_f$$

در اکثر سیستم ها t_m بین 10 تا 200 نانو ثانیه می باشد.

انتساب قاب ها یاframe ها به پردازه ها

- **انتساب مساوی.** در این روش قاب های حافظه اصلی بین پردازه ها به صورت مساوی تقسیم می شود به عنوان مثال اگر حافظه اصلی دارای 12 قاب باشد و 5 پردازه بخواهند به طور هم زمان اجرا شوند به هر کدام از پردازه ها دو قاب تخصیص می یابد و دو قاب باقی مانده به عنوان قاب های بافر در نظر گرفته می شود. پس اگر n قاب داشته باشیم معمولا k تا از این n قاب را به عنوان بافر در نظر گرفته وn-k قاب باقی مانده را مابینm پردازه تقسیم می کنیم.
- **انتساب متناسب.** در این روش تعدا قاب ها متناسب با اندازه پردازه ها به پردازه ها تخصیص می یابد به عنوان مثال اگر حافظه اصلی دارای 12 قاب باشد و دو قاب از آنها به عنوان بافر در نظر گرفته شود در صورتی که اندازه هر قاب 1kباشد و در

سیستم سه پردازه $2k$, $3k$, $5k$ به طور همزمان در حال اجرا باشند به پردازه $5k$، 5قاب، به پردازه $3k$، 3قاب و به پردازه $2k$، 2قاب در نظر گرفته می شود.

- هیچ کدام از دو روش قبلی الویت را در نظر نمی گیرند می توان کاری کرد که به پردازه ها به نسبت الویتشان قاب تخصیص داد.

در هر مراجعه به حافظه اعمال زیر رخ می دهد

در روش مدیریت حافظه مجازی به آدرس منطقی آدرس مجازی می گویند اگر بیت v یک باشد بدین معناست که page مورد نیاز در حافظه اصلی موجود است اگر صفر باشد دو برداشت از آن وجود دارد.

1- چنین page ی اصلا وجود ندارد یعنی دسترسی غیر مجاز است به عنوام مثال پردازه شما دارای سه page است ولی شما مراجعه به page چهارم را می دهید که این دسترسی غیر مجاز است.

2- چنین page ی وجود دارد ولی در حافظه اصلی نیست.

در روش حافظه مجازی به طریق صفحه بندی نیازی، در هر مراجعه به حافظه مراحل زیر دنبال می شود.

1- ابتدا به ساختار جدل صفحه مراجعه می شود اگر صفحه مورد درخواست در حافظه اصلی باشد شماره قاب از ساختار جدول صفحه بدست آمده بنابراین آدرس فیزیکی آماده می شود.

2- اگر بیت v=0 باشد و صفحه مراجعه شده در پردازه وجود نداشته باشد، دسترسی غیر مجاز است و پایان پردازه است

3- اگر بیت v=0 باشد ولی صفحه مورد نظر جزء پردازه باشد دراین صورت خطای فقدان صفحه رخ داده و بنابراین

الف. در حافظه اصلی قاب خالی وجود داشته باشد، صفحه مورد نظر از رسانه جانبی به حافظه اصلی منتقل شده و جدول صفحه update می شود.

ب. در حافظه اصلی قاب خالی وجود ندارد از این رو یکی از صفحات موجود در حافظه اصلی مطابق الگوریتم های جایگزینی به رسانه جانبی منتقل شده و صفحه مورد نیاز از رسانه جانبی به حافظه اصلی منتقل می شود.

برای این که انتقال صفحه مورد نیاز از حافظه جانبی به حافظه اصلی منتظر خالی شدن قاب حافظه اصلی نباشد، صفحه مورد نیاز از حافظه جانبی به یکی از قاب های بافر منتقل شده و بنابراین عمل ورودی و خروجی سریعتر انجام میشود. سپس بعد از انتقال قاب شایسته ی خروج از حافظه اصلی به حافظه جانبی،قاب خالی شده به لیست قاب های بافر اضافه می شود.

الگوریتم های جایگزینی صفحه

مساله الگوریتم های جایگزینی صفحه یعنی هنگامی که نیاز است صفحه ای را از حافظه خارج کنیم کدام یک را انتخاب کنیم تا کارائی حداکثر شود. در زیر الگوریتم های متعددی را برای جایگزینی صفحه شرح می دهیم. ولی به طور کلی الگوریتمی بهتر است که تعداد نقض صفحه های(page fault rate) آن کمتر باشد بدیهی است که هر چه قدر تعداد فریم های آزاد بیشتر باشد، تعداد خطا های صفحه ای کاهش می یابد.

1- الگوریتم FIFO : ساده ترین روش جایگزینی صفحه است. در این روش صفحه ای برای خروج انتخاب می شود که زودتر از بقیه صفحات وارد حافظه اصلی شده باشد

مثال. مراجعات (از چپ به راست) 4 , 3 , 2 , 1 , 0 , 4 , 1 , 0 , 3 , 2 , 1 , 0 را با سه قاب آزاد در نظر بگیرید در روش جایگزینی FIFO چند نقض صفحه رخ می دهد.

تقاضا	0	1	2	3	0	1	4	0	1	2	3	4
قاب 1	0	0	0	3	3	3	4	4	4	4	4	4
قاب 2		1	1	1	0	0	0	0	0	2	2	2
قاب 3			2	2	2	1	1	1	1	1	3	3
نقض صفحه	*	*	*	*	*	*	*			*	*	

ملاحظه می شود که در این مثال 9 بار خطای نقض صفحه رخ می دهد.

ناهنجاری بلیدی: اگر در الگوریتم جایگزینی با افزایش تعدادقاب ها(Frame) تعداد خطای صفحه افزایش یابد گوئیم آن الگوریتم دارای ناهنجاری بلیدی است که الگوریتم FIFO یکی از آنها می باشد.

ادامه دارد.....